In compliance with the
Canadian Privacy Legislation
some supporting forms
may have been removed from
this dissertation.

While these forms may be included
in the document page count,
their removal does not represent
any loss of content from the dissertation.

University of Alberta

Supervisory Control of Multi-Robot Systems

by

Marc Perron

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

Department of Computing Science

Edmonton, Alberta
Fall 2003

National Library of Canada
Bibliothèque nationale du Canada

Acquisitions and Bibliographic Services
Acquisisitons et services bibliographiques

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 0-612-87859-7
Our file Notre référence
ISBN: 0-612-87859-7

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou aturement reproduits sans son autorisation.

Canada

University of Alberta

Library Release Form

Name of Author: Marc Perron

Title of Thesis: Supervisory Control of Multi-Robot Systems

Degree: Master of Science

Year this Degree Granted: 2003

Permission is hereby granted to the University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves all other publication and other rights in association with the copyright in the thesis, and except as herein before provided, neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material form whatever without the author's prior written permission.

Date: 21/07/2003

University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **Supervisory Control of Multi-Robot Systems** submitted by Marc Perron in partial fulfillment of the requirements for the degree of **Master of Science**.

_____

_____

_____

Renee Elio

Date: July 14, 2003

Abstract

Telerobotics and supervisory control allow complex tasks to be performed at large distances or in hazardous environments without posing any real risk to the human operating the system. The obvious advantage of this stems from the capabilities of the human reasoning system which are far superior to any artificial intelligence in existence. The disadvantage is that the communication channel between the work site and control site may limit the amount of sensory feedback available to the operator, often making it difficult to use this exceptional reasoning ability to its fullest extent. This disadvantage can be alleviated by placing intelligent functions at the work site, eliminating the need for the operator to excercise fine control of the robots. Additionally, a single effector may not be enough to perform certain tasks. For example, a telerobot at the work site may not be able to move objects larger than itself without help. By introducing more robots, and having them cooperate to accomplish a task, the productivity of the overall system should increase dramatically. However this may have the adverse affect of decreasing the productivity of the operator as he would need to control the whole group of robots at the work site. This dissertation examines the challenges faced by a human who is operating or supervising a system of multiple tele-autonomous robots and proposes some sollutions to simplifying their task. A system of simulated, soccer playing robots will serve as an example to illustrate these various challenges. The simulation environment consists of the RoboCup Soccer Simulation Server. The team of robots, team SimCanuck, operates within this simulated environment using a reactive controller based on the subsumption architecture. The on-line coaching functions provided by the simulation environment are used as the conduit to allow a human coach to communicate strategy information to the team which should be accomplished cooperatively. A comparison of two identical systems is performed to measure the relve abilities of a coached team versus an un-coached one. Relative goal differential is used as a performance metric to demonstrate statistical superiority of the supervised multi-robot system.

Contents

1 Introduction 1
- 1.1 Motivation . . . 1
  - 1.1.1 Will a MATS Benefit from Both its Parent Technologies? . . . 4
  - 1.1.2 Will a MATS Prove More Effective Than an Identical Autonomous System? . . . 4
- 1.2 Structure of this Dissertation . . . 5

2 Collective Robotics 6
- 2.1 Languages, Communication and Representation . . . 7
- 2.2 Cooperation and Organisation . . . 9
  - 2.2.1 A Game of Politics and Philosophy . . . 10
    - 2.2.1.1 Federated Systems . . . 11
    - 2.2.1.2 Hierarchical-Authoritarian Systems . . . 11
  - 2.2.2 Communication . . . 12
    - 2.2.2.1 Communication in Biological Systems . . . 13
- 2.3 Autonomous Agents . . . 15
  - 2.3.1 Intentional Systems . . . 16
  - 2.3.2 Tropistic and Hysteretic Agents . . . 18

3 Telerobotics, Teleautonomy and Supervisory Control 22
- 3.1 Some Definitions . . . 24
- 3.2 Classifications of Telerobotic Systems . . . 25
- 3.3 Collaborative Control . . . 27
- 3.4 Human Operator Control Interface . . . 28
  - 3.4.1 Theories of Interaction . . . 30

4 The SimCanuck Tele-Autonomous Soccer System 32
- 4.1 Semi-Autonomous Soccer Agents . . . 32
  - 4.1.1 Behaviour Controller . . . 33
    - 4.1.1.1 Behaviour Hierarchy for the Forwards, Mid-Fielders and Defenders . . . 34
    - 4.1.1.2 Behaviour Hierarchy for the Goalie . . . 38
  - 4.1.2 Missions, Tasks, Plans and Actions . . . 40
  - 4.1.3 Plan Generation . . . 45
- 4.2 The Supervisory Coach Component . . . 46
  - 4.2.1 Filtering Control Information . . . 47
  - 4.2.2 Coach Graphical User Interface . . . 48
    - 4.2.2.1 The *Play-Book* Operating Mode . . . 48
    - 4.2.2.2 The *Coach* Operating Mode . . . 48

5 Actions and Behaviours 51
5.1 Basic Definitions . . . 52
5.2 Modeling the Subsumption Network . . . 54
5.2.1 The Notion of Subsumption . . . 55
5.2.2 Using Outputs as Inputs . . . 58
5.2.3 Using Multiple Inputs and Outputs . . . 59
5.3 Specifying and Reasoning about Multi-Agent Systems . . . 61
5.3.1 Actions and their Effect on the Environment . . . 61

6 Analysis and Results 63
6.1 Empirical Analysis of the SimCanuck MATS . . . 64
6.1.1 Qualitative Analysis of the Supvervised SimCanuck MAS . . . 67
6.2 Theoretical Analysis of the SimCanuck MATS . . . 70
6.2.1 Modeling the SimCanuck Behaviour . . . 70
6.2.2 Toy Example Using a 4 × 4 Grid World with Tropistic Agents . . . 76
6.2.3 Toy Example Using the 4 × 4 Grid World with Hysteretic Agents . . . 81
6.2.4 Using Hysteretic Agents with Missions Issued at Run-Time . . . 88

7 Conclusion 91
7.1 Contributions . . . 95
7.1.1 The SimCanuck MATS Framework . . . 96
7.1.2 Logic Formalism for Subsumption Controllers . . . 96
7.1.3 Experimental and Theoretical Results . . . 97
7.2 Future Work . . . 98

Bibliography 100

A RoboCup Soccer Simulation Server 106
A.1 Soccerserver . . . 106
A.1.1 Client Command Protocol . . . 106
A.1.2 Sensor Model . . . 108
A.1.2.1 Auditory Sensor Model . . . 108
A.1.2.2 Visual Sensor Model . . . 108
A.1.2.3 Body Sensor Model . . . 109
A.1.3 Physics of the Simulation Environment . . . 109
A.2 Online Coach Client . . . 109
A.2.1 On-line Coach Command Protocol . . . 111
A.2.2 Modifying the Behaviour of the Coach . . . 111

B CLang Syntax 113

List of Tables

3.1 Relative capabilities of humans and machines. *Source* [82]. 29

6.1 Results of the sample games played between two unsupervised instantiations of the SimCanuck robotic soccer system. 65
6.2 Results of the sample games played between a supervised instantiation of the SimCanuck robotic soccer system and an unsupervised one. 66
6.3 Listing of the probabilities of the final positioning of robots $(r, 1)$ and $(r, 2)$ after six steps using a simple subsumption based controller. 81
6.4 Listing of the probabilities of the final positioning of robot$(r, 1)$ and robot$(r, 2)$ at state s_6 using a simple subsumption based controller in hysteretic agents with a complete mission specification. 86
6.5 Probability of successfully completing the mission μ_2 in a given number of steps. 87

7.1 Properties of tele-autonomous and multi-robot systems 92

A.1 Soccerserver parameters for agent auditory sensors. 108
A.2 Soccerserver parameters for agent visual sensors. 110
A.3 Soccerserver parameters for the body sensor model. 110
A.4 Soccerserver parameters for the coach client. 112

List of Figures

2.1 Representation of the Functional Components of an Autonomous Agent [34]. 17

3.1 Spectrum of telerobotic interaction models. 22
3.2 Matrix of agency vs. autonomy in telerobotic systems 27

4.1 A diagrammatic view of the components of the SimCanuck robots organised into a "cortex" like structure. 33
4.2 A diagrammatic representation of the subsumption architecture of the behaviour controllers for robots fulfilling the forward, defender or mid-fielder roles. 34
4.3 A diagrammatic representation of the subsumption architecture of the behaviour controller for the robot fulfilling the role of the goalie. 38
4.4 Active behaviour modules during the execution of a plan that involves dribbling the ball to a specified point (illustrated in gray) and the flow of signals between these modules (bold signal lines). 42
4.5 Specification of a *cross play* mission using the on-line coach protocol defined for the RoboCup Simulation Server. 44
4.6 Screen shots of the coach's graphical user-interface. 49

5.1 Diagram of a subsumption controller that performs operations on strings. . . 60

6.1 Game six ball movement overview. 69
6.2 Game seven ball movement overview. 69
6.3 A simple behaviour network for the grid-world robots. 72
6.4 Examples of the execution of the *chase ball* behaviour in a grid $\mathcal{G}_{4,4}$ with a single robot and a single ball. 76
6.5 Initial state of the example grid world with a 4×4 grid, 1 ball object and 2 robots belonging to the same collective. 76
6.6 Probabilities of robot positioning at final state s_6 using a simple subsumption based controller. 82
6.7 Mission specification using the RCSS coach language for moving the ball to position $< 4, 1 >$ in a grid world instance $\Gamma^{1,2}_{4,4}$. 84
6.8 Probabilities of robot$(r, 1)$ reaching position $< 4, 1 >$ between 4 and 1000 steps. 88

Chapter 1

Introduction

The idea of telematics was first conceived by Simon Nora and Alain Minc as "a broad field of computer-mediated communications, such as the Internet" [79]. Through these computer-mediated communications, new kinds of human-human and human-machine interactions have become possible. Telepresence, teleconferencing and telerobotics are all enabled by the mechanisms encompassed by telematics. Of special interest for this dissertation is the idea of telerobotics and human supervisory control. These techniques allow complex tasks to be performed at large distances or in hazardous environments without posing any real risk to the operator. Tele-autonomous systems, an extension of telerobotics, are the result of merging these techniques with those of autonomous systems. This increases their effectiveness by giving them the ability to make local decisions that would otherwise be difficult for a remote operator.

At the work site, an autonomous or semi-autonomous system is responsible for carrying out the tasks set out by the operator at the control site. This system will often take the form of a mobile robot with an array of manipulators for performing various tasks. Having a single mobile robot, however, implies a single point of failure. If the system is intended to work in hazardous environments, a mechanical failure in the mobile robot may prevent the completion of the task and possibly result in the loss of expensive hardware on the mobile robot. Multi-robot systems provide a means of fielding a group of simple mobile robots with greater flexibility and robustness.

1.1 Motivation

The union of telerobotics and multi-robot systems, termed as multi-agent telerobotics by Subhi Ali [89], may potentially yield systems that combine the power and robustness of both. Each technology possesses desirable properties which would be most beneficial if combined to form a multi-agent telerobotic system (MATS).

When compared to single autonomous robots, multi-robot systems provide advantages

by: allowing a greater range of possible tasks, increased efficiency for some tasks, increased reliability, decreased cost and greater ease of development. Theoretically, any task that can be accomplished by a single robot can also be accomplished by more than one. Conversely, multiple robots should be able to accomplish many tasks that are impossible for a single agent. For example, any task that requires coordinated actions at a distance is most likely spatially impossible for a single robot. The range of tasks that are possible for a group of robots is therefore a superset of those possible by a single one.

A robot collective may also provide performance advantages over single agent systems. For example, foraging and grazing tasks are more efficiently accomplished by a group of robots. Evidently, tasks that require the complete coverage of the work environment will be more quickly accomplished if each agent in a group covers its own disjoint area; the greater the number of members in a collective, the more area of the environment can be covered concurrently.

The reliability of a system is also increased if a system is composed of more than one robot. As stated previously, a single autonomous robot implies a single point of failure that could prevent the completion of a given task. Conversely, if a system is composed of many robots, even if one or a few were to be damaged preventing them from fulfilling their role, the joint task may still be completed by the balance of the robots in the collective. It may also be possible for the functional robots to collect damaged ones and return them to the common extraction point so that they can be repaired or some of their parts salvaged.

This question of robot failure also raises issues of cost. A single multi-function robot may be much more expensive to produce than a group of simpler heterogeneous ones. This initial cost may also be compounded by the cost of losing the robot if one of its components might fail rendering it unable to complete its task and return. The cost of losing one or more robots in a collective will most likely be much less than the loss of a single monolithic, multi-function robot. Additionally, because each robot in the collective will in most cases be significantly less complex than their multi-function counter part, their development time may be reduced. This is especially true when replacing robots that may have been lost in a previous mission. If only a single robot were deployed, either it can be re-used in the future or it must be replaced; conversely, if members of a collective are damaged or lost, only those need to be replaced for future missions.

Telerobotics also has several desireable properties: they provide greater flexibility, the possibility to act on unexpected opportunities, greater social acceptance and the possibility for a system to learn from human experts. It is a well established fact that humans are more capable in accomplishing general tasks than their robotic counterparts. The human visual system is unparalleled by any computer vision or digital signal processing system. The human cognitive process also far exceeds the capabilities of even the most advanced reasoning

algorithms in existence. The strength of computers rather lies in number crunching and analysis of large volumes of digital information. Allowing a human to guide an autonomous or semi-autonomous system allows it to more effectively cope with its shortcomings compared to the human cognitive process and sensory system. Conversely, the autonomous or semi-autonomous system's computational capabilities can be advantageously used to shield the human operator from the analysis of voluminous digital information and sheer number crunching tasks which are more suited to digital computer devices. For example, an automated system might be successfully used in an air traffic control tower to route planes as route-allocation algorithms are fairly well understood; in high volume traffic situations, this kind of tasks lends itself better to machine capabilities than human ones. However, in the event of some kind of emergency where aircraft must quickly be re-routed and their arrival schedules changed, a human air traffic controller becomes a vital part of the whole system. The reasons for a given emergency may be unknown to an automated system therefore limiting its ability to deal with unknown situations in a flexible and timely manner.

During its operation, an autonomous system may encounter an event or an obstacle in the work environment which is unexpected and thus it may not know how to effectively deal with it. Allowing a human operator to intervene when the autonomous system does not have a plan to deal with these events allows it to be more flexible in dynamic environments and provides a means for the encountered obstacles to be overcome. This is especially evident in adversarial environments such as, for example, robotic soccer; it logically seems beneficial to have a human operator to deal with highly variable and unpredictable situations brought about by the uncertainty in opponent actions.

A robotic system under the control of a human operator will also, in many cases, be more socially accepted due to the distrust many people have of technology. People will generally be more comfortable with a system that allows a human to participate in the control loop; especially if these can potentially put lives at risk. For example, in the motion picture Wargames, a government military computer nearly initiates global thermonuclear war thinking it is a game; military officials were unable to stop it as the system was designed to function without human input.

Autonomous systems are designed and constructed by humans, therefore it is natural to say that their capability to solve a particular problem will only be as good as the designer's understanding of it. Because humans are not omniscient, neither should their machines be; therefore it logically follows that a system designed by humans may yet learn from them as even humans will generally learn something new on a daily basis.

1.1.1 Will a MATS Benefit from Both its Parent Technologies?

The properties of multi-robot and tele-autonomous systems provide clear gains in their respective areas. So naturally one would think that a system that inherits from both of these technologies would also benefit from these advantages. A multi-agent telerobotic system should provide the benefits of both multi-robot and tele-autonomous systems. However, one important question that needs to be answered is wether the implementation of the properties of one technology might not interfere with those of the other. For example, if a designer wants to build a multi-robot tele-autonomous system that will learn from expert operators, the task of implementing a learning algorithm for a collection of robots will be much more complicated than for a single robot; maybe even prohibitively so. Conversely, if the system is designed with a single robot to allow for the learning capabilities to work, the robustness provided by a multi-robot system is lost. The compromise would be to field a multi-robot system with a one to one correspondence of expert human operators and robots. This may not be feasible however as the availability of human experts may be limited. System designers must therefore be carefull when building multi-agent telerobotic systems to ensure that the desired properties of each of its parent technologies are retained by the new system.

1.1.2 Will a MATS Prove More Effective Than an Identical Autonomous System?

In many cases, the robots at the work site of a tele-autonomous system will be fully autonomous and only using an operator's commands when these are available. Conversely, the human operator may let the worker robots accomplish their assigned task autonomously until such a time when their input is required. If a multi-agent telerobotic system is used in an adversarial environment, there will be other agents trying to prevent it from achieving its goals. In the robotic soccer domain, there will be two teams of robots playing against each other. In this case, fielding a team of autonomous robotic soccer players that can receive instruction from a human coach should be at an advantage over a purely autonomous opponent; this would be a good indication that the properties of telerobotic and multi-agent systems have successfully been inherited by the multi-agent telerobotic one. The question remains however, wether a tele-autonomous team of robotic soccer players have an advantage over an identical autonomous system. In this context, two systems will be identical if, given the same situation, they make the same "decision" when randomness and supervisor advice are ignored.

1.2 Structure of this Dissertation

The goal of this research is primarily to attempt to answer the questions in section 1.1.1 and section 1.1.2. These questions will be answered in verifying the following hypothesis:

> Given a system of simple mobile robots that are capable of interpreting and acting upon *just-in-time* task specifications, this system will exhibit desirable properties of both tele-autonomous and multi-robot systems, making it better equipped to deal with dynamically changing environments, that predicate dynamically changing short term goals, than an equivalent system of purely autonomous robots.

Using a simulated robotic soccer environment as a vehicle, arguments and evidence will be put forth to support this hypothesis. The structure of the rest of this dissertation will be as follows. Chapter Two will provide an overview of the current research in collective robotics and multi-agent systems. Much of the work presented in this chapter has served as the ground work for developing the SimCanuck robotic soccer team that was used as the experimental model from which observations will be drawn to support the hypothesis above. Similarly, Chapter Three will give an overview of current research in telerobotics, tele-autonomous systems and human supervisory control. This work provided a source of inspiration for the supervisory control system implemented for the SimCanuck robots. Chapter Four discusses abstract implementation details of the SimCanuck control architecture and supervisory control framework. In Chapter Five, a theoretical model based on an interpretation of a first order formal language is defined and discussed. The discussion will help to support the hypothesis given above. A discussion of experimental observations is also given in this chapter. Finally, Chapter Six provides conclusions about the evidence found in the discussion in Chapter Five which supports the hypothesis for this work. Future possible directions for continued research are also presented.

Chapter 2

Collective Robotics

Collective robotics is a field of research that studies robotics as a society. Groups of robots can be cobbled together and in many cases are able to perform tasks that single robots cannot. Collective robotic systems typically involve multiple embodied, intelligent agents that are social in character where the properties of the global system will not necessarily be deriveable solely on the basis of the abilities and properties of its component robots. To illustrate the advantage of multi-robot systems, groups of robots should be able to move objects far too large for any one robot to move on its own. In these instances, the question of teamwork becomes extremely important.

There are many problems faced by autonomous multi-robot systems that are not considered for single autonomous robots. These problems are:

1. *How to formulate, describe, decompose and allocate tasks among a group of intelligent mobile robots.* Some approaches to this problem involve negotiation among members of the society to establish joint commitments [24]. Other approaches, more suitable in time-critical environments where communication is restricted, prescribe the synchronisation of team goals during periods of derestricted communication [87]. The formulation problem is also faced by researchers in Distributed Artificial Intelligence (DAI) in the form of problem decomposition. Bases such as abstraction levels, functional, data or control dependencies and interaction density are among the suggested solutions to these problems in DAI [36].

2. *How to enable agents to communicate and interact.* This problem deals with the communication languages or protocols to use as well as when agents should communicate and what the contents of their messages should be. Cohen, Levesque and Smith assume the use of performative utterances, or self-referential auditory communications, for the act of communication between agents [24]. They define communicative actions such as requests, assertions and acquiescence (accede) as a means of establishing mutual beliefs between agents. Natural language can also be used for communication but

due to the complexity involved in interpreting natural language computationally, this solution is generally infeasible.

3. *How to ensure that agents act coherently in making decisions or taking action.* Primary approaches to solving this problem include the establishment of organisation [42, 43], multi-agent planning [55], abstraction and the establishment of joint commitments to actions or plans [24]. One of the potential pitfalls faced by solutions to this problem is the subtle difference between acting together and acting as a coordinated unit. A group of agents will be acting together if they all have a mutual goal which they are trying to satisfy individually. Coordinated action requires a joint goal – a goal common to all the agents – that they have established and will satisfy cooperatively. For example, agents may have mutual goals and yet will only set out to accomplish these on an individual basis (e.g. taking cover from the rain). If the agents were working together to achieve a common goal, this would constitute coordinated behaviour.

4. *How to enable agents to represent actions, plans and knowledge of the other agents in the society.* The ability to reason on other agent's actions, plans and knowledge is a useful skill when trying to coordinate actions of the society. The establishment of a social organisation allows certain rules pertaining to agent-agent and agent-environment interaction to be enforced providing a basis for agents to make assumptions regarding the beliefs, desires and intentions of their peers. These social organisations represent a framework through which coordination can be realised.

Clearly multi-robot cooperation offers advantages for achieving certain tasks that would be impossible for uninominal or adversarial systems. They offer the possibility of "enhanced task performance, task reliability and decreased cost over more traditional robotic systems"[31]. However, as stated above, many difficulties arise when distributing work among actors.

2.1 Languages, Communication and Representation

It's impossible to build computer programs without the use of languages to represent, describe and implement the required structures and processes. Similarly, the design of multi-agent systems involves the use of a large number of languages and formalisms to describe all of their aspects. The languages of multi-agent systems can be classified into five major types: implementation languages, formalisms to describe multi-agent systems, languages for the representation of agent's knowledge, for the definition of their behaviours and for their communication. These five types of languages will be elaborated in more detail subsequently.

The importance of the languages used to describe multi-agent systems stems from the benefits they provide to the designer of such systems. The list of potential benefits in-

clude an increased capacity to express ideas, improved backgrounds for choosing appropriate languages and better understanding of the significance of specific implementations. The following is a taxonomy of languages which was outlined by Jacques Ferber [34].

Language type L1: Implementation languages

Implementation languages are useful for the actual implementation of multi-agent systems. These languages may include functional languages such as COMMON LISP, ML and Miranda, logic languages such as Prolog or Object oriented languages such as C++ and Smalltalk. The various implementation languages, with their various semantic differences, provide the multi-agent system designer with an array of tools to solve a variety of problems. These languages express all of the details of the inner workings of each component in the system. For this reason, they are not typically well suited as an abstract representation of the system (with the possible exception of Smalltalk). This class of languages is meant to be representational, or concrete, rather than abstract.

Language type L2: Communication languages

The second type of language is used to describe the exchanges between the various agents in a multi-agent system. These help ensure that interactions between communicative agents, for the pruposes of transmitting information or mutual requests for information or services, is possible. In fact, communication languages can serve as the *glue* that will eventually allow heterogeneous agents to coordinate their actions, to cooperate in accomplishing a joint goal or to integrate themselves into a group of agents. This class of languages is more abstract than implementation languages yet still must provide enough of the detail to be practical as a communication language.

Language type L3: Behaviour description languages.

To directly describe the behaviours of agents or the laws that regulate their environment using implementation languages introduces needless details that mask essential elements of the system. For this reason, more abstract representations and formalisms are needed to describe the agents and their environments. This technique is used in the area of distributed computing to describe the services provided by software objects in an implementation independent way. For example, the Common Object Request Broker Architecture (CORBA) defines the interfaces of software objects using a common Interface Definition Language (IDL). This describes the operations that are made available by objects without revealing the implementation language or the operating platform in use. Similarly, it is beneficial to develop formalisms to describe the behaviours of agents without the need to settle on an actual implementation language. Examples of such formalisms include rule based languages, finite state automata, turing machines and petri nets.

Language type L4: Languages for the representation of knowledge.

Languages to represent knowledge are mostly used by cognitive agents that have to perform some deeper reasoning about the environment in which they operate. Using these languages, agent's can construct internal models of the environment in which they evolve and work. This can provide them with the necessary information to reason about their environment and make predictions based on their current knowledge. This class of languages is usually associated with logic based systems. They can be used not only to model the "mental" state of the agent by representing their beliefs and goals, but also as a means of describing an agent's behaviour. This aspect of the fourth language type overlaps with the behaviour description languages.

In reactive agents, the representation of knowledge is not so important as they do not, by definition, perform much reasoning about their environment. The use of this language type becomes imperative however, when dealing with more complex cognitive agents. Most artificial intelligence oriented languages are found in this class. Rule based and blackboard based languages, and structured knowledge representation languages such as semantic networks or frames are members of this class.

Language type L5: System specification languages.

The final level of language abstraction contains languages that can be used, on the one hand, to formalise what is understood by a multi-agent system using the notions of interaction and the concept of intention. On the other hand, languages of this class can also be used to specify the conditions that must be respected when modeling and implementing these systems. These 'meta' languages will generally have foundations in mathematics and are often used to reason about the other classes of languages. These provide multi-agent system designers with highly abstract tools to represent the overal system.

The five classes of languages described previously are not mutually exclusive. It is possible to find languages that are members of two or more classes. Combinations of languages from each class may also be used to represent components of a system. For example, a communication protocol for inter-agent communication may use a language of type L2, a communication language, as a vehicle for conversation between agents and a language of type L3, a behaviour description language, to describe the mechanisms for interpreting the messages.

2.2 Cooperation and Organisation

The essence of team behaviour is cooperation; if a team has a goal and a plan to achieve that goal, team members are not only individually committed to the success of their portion of the plan, they are also committed to the success of

the overall team goal, and to the success of the other team members achieving their portions of the overall joint plan. [24]

When taking on the challenge of cooperation in a multi-robot system, one must ask the question of how to usefully conceptualise planning and action when dealing with multiple participants. Task allocation can be approached by decomposing a single task into sub-tasks and distributing them within the collective. Organisational mechanisms may be used to assist in task allocation (or distribution) in order to perform a single collective task. These mechanisms can have a profound impact on the coordination of organisational processes which can be modelled by a set of organisational interdependencies. Coordination can be considered as an effect of the following classes [65]:

- Workflow interdependencies refer to activities that achieve a task. These can be considered as consumer/producer dependencies where some activities will consume something that was produced by another activity; therefore certain pre-conditions may be required before undertaking certain activities.
- Process interdependencies which occur between activities that are part of different work flows. These dependencies are present when multiple activities can be performed concurrently and must access common resources or must be synchronised properly.
- Interdependencies of scale that are present between similar activities that might be more efficient if they were grouped together.
- Social interdependencies comprising subjective factors such as social needs or different personalities that determine the relation between workers. For artificial agents, these dependencies will result from how each agent was designed; some agents may be more suitably equiped to perform certain tasks than others.

This distribution can be managed in either a centralised or decentralised way which will necessitate different approaches to organising the system.

2.2.1 A Game of Politics and Philosophy

Despite their applicability to multi-agent systems and robotics, the study of organisational structures has predominantly been in the realm of philosophy, psychology, biology and politics. Organisational systems can be analysed by relating them to models of complex activities of human and animal societies [49]. "Some animal societies such as colonies of ants and bees, flocks of birds, schools of fish, can be an inspiring model for a self-organizing robotic network" [63].

Human organisations can also serve as a model for organising robot collectives. Social systems can "exhibit global behaviours that enable them to solve problems that would

confound any of their individual members"[54]. Of particular interest for this work are the concepts of federated and authoritarian organisations which represent decentralised and hierarchical systems respectively.

2.2.1.1 Federated Systems

There are two forms of federated organisational systems: *political federalism* and *social federalism*. Political federalism is a bottom-up organisation into a form of centralism resulting from the formation of small groups within the collective while social federalism looks for a balance between individualism and collectivism [45]. Federated organisations follow a honeycomb structure where decisions are made at a communal level by small groups in a collective. For example, "although no individual could accurately predict the number of loaves of bread to be consumed in New York City on a given day, the entire system of New York bakeries does an excellent job of keeping the city stocked with bread"[54]. This illustrates an emergent global behaviour of keeping New York City stocked with bread.

Global behaviour in federated systems may either emerge from the interactions of the various associations that constitute it; or be agreed upon by them. In this context, an association describes one of the parties of a barter agreement, the other party being an individual who binds himself to the association [51]. In the example cited above, the interactions between associations are reduced to the notion of competition. Bakeries remain profitable by selling as much bread as they can produce and not producing more than they can sell. As each bakery is in competition with every other bakery, they try as hard as possible to maintain profitability, and thus keep New York City stocked with bread with minimal waste.

A robot collective can be modelled as an association where each member of the association has agreed to adopt the joint intentions of the group. This idea was proposed by Cohen, Levesque and Smith in their characterisation of joint intention:

Joint Intention x and y jointly intend to do some action iff it is mutually known between x and y that they each intend that the collective action occur, and also that they each intend to do their share as long as the other does likewise, and this mutual knowledge persists until it is mutually known that the activity is over (i.e., is successful, unachievable, irrelevant) [24].

The formation of such an association allows us to assume that if both parties does their share, joint action will succeed. The global behaviour will then emerge from their interactions.

2.2.1.2 Hierarchical-Authoritarian Systems

An authoritarian organisation is characterised by a top-down hierarchy. Decisions are passed down from the top toward the bottom and the lower layers in the hierarchy are obligated to

respect the decisions handed down from above. This kind of organisation can be observed in human institutions such as the military or in the corporate world. The military has a chain of command that is strictly respected and must be for its effective operation. Corporations are also hierarchically organised with the shareholder at the top and labourers at the bottom. This organisational model follows a spider-web structure where the middle is supported by the extremities.

Task allocation in a hierarchical organisation of this sort is performed in a centralised manner via pre-defined mediators. The simplest case involves a single mediator for the entire agent society. The mediator has a table of its acquaintances which stores information about their capabilities. This type of mediation is also possible in the honeycomb model. However in the hierarchical model, task allocations are imposed as is the case in authoritarian social organisations.

The benefits of using centralised allocation in an top-down hierarchical society of agents are: efficiency of coordination, quick availability of up-to-date status information of the collective and ease of satisfying optimisation goals [34]. Efficiency in the coordination of tasks is a clear benefit of this paradigm as no negotiation is required in the task allocation process. However, hierarchical organisation is predicated by a *master-slave* relationship which, although often allows for more efficient conduct, introduces brittleness to the system which is characterised by the small number of sources of guidance for other cooperating robots [6].

2.2.2 Communication

Communication is an important part of multi-robot cooperation. In many cases, cooperation and communication are tightly coupled; communication between agents is vital for effective cooperation, and failures lead to lapses in coordination. This can be illustrated using the principle of the *prisoners' dilemma*. The prisoners' dilemma occurs when social entities must simultaneously choose an action without knowing what the others will decide. The fact that entities do not know what actions are being taken by others in the collective is a result of their inability to communicate.

Communication can take place either directly (explicit) or indirectly (implicit). Direct communication involves an explicit exchange between agents; this is impractical in situations where the range and bandwidth are limited. Also direct communication schemes may be subject to failure if the topology is either tree-like or address-based. Tree-like communication topologies imply a hierarchy, and failure in a particular robot will result in the isolation of the robots on either side of the node where the failure occurred [31]. If address-based communication is used, distinct roles may need to be assigned to the robots thus restricting their interchangeability resulting in the global system becoming more brittle. Two design

alternatives for explicit communication, described by Dudek et al. [32], are *one-way* and *two-way* communication. One-way communication implies a tree-like communication topology where a single *leader* robot sends signals to *follower* robots [97]. Two way communication does not require such a hierarchical organisation of the robot group. Agents following this communication strategy are in constant duplex communication (i.e. both can transmit and receive).

Balch and Arkin [12] used explicit communication to transmit state and goal information between robots performing foraging, consuming and grazing tasks. Explicit signaling about a robot's state was accomplished by mounting a binary signal atop the robots, visible to other robots, which was either *off*, to indicate the robot was in a wander state, or *on* to indicate that it was in any other state. This communication mechanism, although trivial, requires explicit signaling and can therefore be classified as explicit communication; it does not, however, suffer from problems of interference as is common in explicit communication schemes.

Indirect communication is more subtle than its direct counterpart and can take the form of passive observation of other agents in the collective or the environment in which they work. Physical changes to the working environment can serve as sensory cues for the exchange of information between robots. Although not as efficient as explicit communication, implicit communication is a more robust mechanism. Because implicit communication involves passive sensing modalities, interference does not pose a problem, thus communication protocols can be simplified.

2.2.2.1 Communication in Biological Systems

Biological systems also provide a source of inspiration for communication mechanisms. Biologists analyse the behaviours of animal societies and draw inferences about their social behaviours and how group coordination is achieved. Some of the most studied systems, described by Jung and Zelinsky in [49], are listed below. The following descriptions illustrate how various biological systems effectively cooperate through various means of implicit and explicit communication as well as learned and innate groundings for meaning.

Bacteria: Bacteria have existed on earth for millions of years and have recently been observed to exhibit primitive cooperative behaviour. It has long been understood that bacteria live in colonies. They can communicate using sophisticated chemical signals and engage in altruistic behaviours which benefit the colony. As an example of cooperative behaviour, mycobacteria assemble into multi-cellular structures, known as fruit bodies, which requires them to have some level of coordination. This is accomplished in a number of stages which are mediated by different chemical signal systems. Bacteria emit (explicit communication) and react to signals in a genetically determined

way that enables them to cooperate. The keys to communication between bacteria are the actions of emitting and reacting. Bacteria exhibit the use of explicit communication through their chemical signal system. Interference may therefore be possible if bacteria are emitting contradictory signals to the rest of the collective.

Insects: Social insect societies also provide a system of simple organisms that exhibit cooperative behaviour. The most studied of these systems are societies of ants, termites, bees and wasps. Through the use of pheremones (implicit communication) and patterns of behaviour such as the bee dance (explicit communication), insects are able to accomplish communication with meaning preservation. Communication of goal information is possible in colonies of honey bees. If a bee discovers a rich source of nectar, it can communicate its location with a "dance" which encodes distance and directional information to guide the other bees in the hive to the source [12].

Ant colonies can implicitly communicate goal information via pheremones left on the trail to the goal. If a goal is a particular rich source, more ants will make their way towards the source thus leaving stronger pheremone trails with a result that more ants will be likely to follow that trail. This process has served as a natural metaphor for the ant colony optimisation algorithm which has been applied to the traveling salespersons' problem [29, 30] and load balancing in telecommunications networks [77] and routing [19].

Wolves: These social animals from the Canine family are carnivores that form packs with strict social hierarchies and mating rituals. Cooperation is achieved via observation, vocalisation and pheremones. In particular, wolves observe the actions of the dominant male who, to some extent, directs the hunt. Pheremones (implicit communication), posture (implicit and explicit communication) and vocalisations (explicit communication) serve as the basis for their system of communication where the grounding is partially learnt. As seen here, wolves make use of many forms of communication. Communication through pheremones is apparent when wolves mark their territory. Posture can be classified either as implicit or explicit communication. The mood of a wolf can be observed by traits of its posture that may or may not be deliberately exhibited. A snarling wolf is clearly angry or feels threatened; the act of snarling is an explicit attempt at communicating that to others within visual range. Much like dogs, an unhappy wolf might indeliberately lower its tail which acts as a sign of the wolf's mood. Vocalisations are clearly explicit communication.

Primates: Non human primates interact through passive observation of collaborators (implicit communication) via visual and auditory cues which are interpreted as actions and intentions. Odors, vocalisations, gestures and facial expressions can provide in-

sight of the mental and psychological state of other primates in a social group. Higher primates, such as the orang-utans, are able to represent the internal goals, plans, dispositions and intentions of others.They can construct collaborative plans by acting socially with other primates in their group.

2.3 Autonomous Agents

An agent can be defined as something that perceives and acts [76]. This definition is very general and is applicable to many different kinds of systems. Agents can take the form of purely digital entities such as web-bots used for indexing web-pages on the internet, or embodied physical robots. In either case, the role of the agent is essentially the same, to collect sensory input and to act on the information gathered. Externally, agents can be seen as acting rationally by ascribing them beliefs, desires and intentions; however, the exact mechanisms that agents use for reasoning may not have purposefully been designed that way.

An autonomous agent can be broken down into six functional components: representational, organisational, conative, interactional, productive and vegetative [34]. The interaction of these functional components results in the overt behaviour of the agent. This is very similar to the idea proposed by Marvin Minsky that human behaviour results from the interpretation of many agencies in the brain. The six functional components of an autonomous agent, described in [34], are explained below:

1. The *representational functions* are the set of functions used for modeling the environment and other organisations and the memorisation of the events that have affected the environment. These functions are more evident in cognitive agents than in reactive agents. Cognitive agents maintain an internal representation of their working environment while reactive agents act on sensory stimulus.

2. The *organisational functions* are the set of functions used for planning or, activity management if the agent is part of a group. These functions are responsible for ensuring the execution and allocation of tasks, and are most important when dealing with a multi-agent system. In these systems, task scheduling and allocation becomes much more important to avoid overlapping of tasks.

3. The *conative functions* (from the latin *conari* meaning to prepare to undertake) can be grouped into three sub-groups. The *motivational* functions describe how goals and tendencies are formed. These may originate from internal motivations (needs, desires, drives) or external social pressure. In this context, social pressure relates to the requirements of other agents in a collective. These external social pressures

will only become a factor in the case where the agent is a member of a society or team. The second group of conative functions describes the *constraints* on the agent's possible actions. The set of possible behaviours is limited by norms, inhibitions and prohibitions. The final group consists of the various *decision making* functions used to evaluate goals and to decide on tasks to undertake.

4. The *interactional functions* provide a means with which an agent can interact with its environment and communicate with other agents in its vicinity. In the case of a mobile robot, these functions include the manipulation of the environment via end-effectors. For example, picking up an object with a robotic hand or rolling to another point in the environment constitutes interactional functions.

5. The *productive* or *operative* functions are concerned with the primitive actions that must be enacted in order to accomplish a task or to solve a problem. These functions are dependent on the task domain. For example, if the agent is required to fulfill linguistic functions, then lexical analysis mechanisms are required.

6. The *vegetative* or *preservation* functions are concerned with maintaining the working state of an agent. For example, if the agent consists of a mobile robot that is responsible for its own battery management, then the vegetative functions will be responsible for this task.

Similar to the nervous, circulatory, reproductive and digestive systems in living beings, the functional components described previously work together to maintain the functionality of the global system; however they are not necessarily actual physical components in an agent architecture. They can be used as abstract modules of the functions of the agent. Figure 2.1 shows the schematic representation of the six functional components of an agent. This figure was presented in [34] both as a representation of a single agent and multi-agent system.

2.3.1 Intentional Systems

One popular formalism for representing robot control systems is the *Belief, Desire, Intention* approach. These types of systems ascribe intentionality to a system and as such, modeling the information it possesses as its *beliefs*, and its goals and subgoals as *desires*. The *Belief, Desire, Intention* (BDI) formalism will generally represent *intentions* as plans, sub-plans or as the deeper reasoning required to make the requisit transitions to the desired state. In this instance, the term *intentional* is taken to mean "by conscious design or purpose."

Intentionality is used mostly as a means of explaining the behaviour of systems via external observation with no real knowledge of the underlying mechanisms it utilises. Daniel Dennett describes intentional systems as "a system whose behaviour can be – at least sometimes – explained and predicted by relying on ascriptions to the system of beliefs and desires

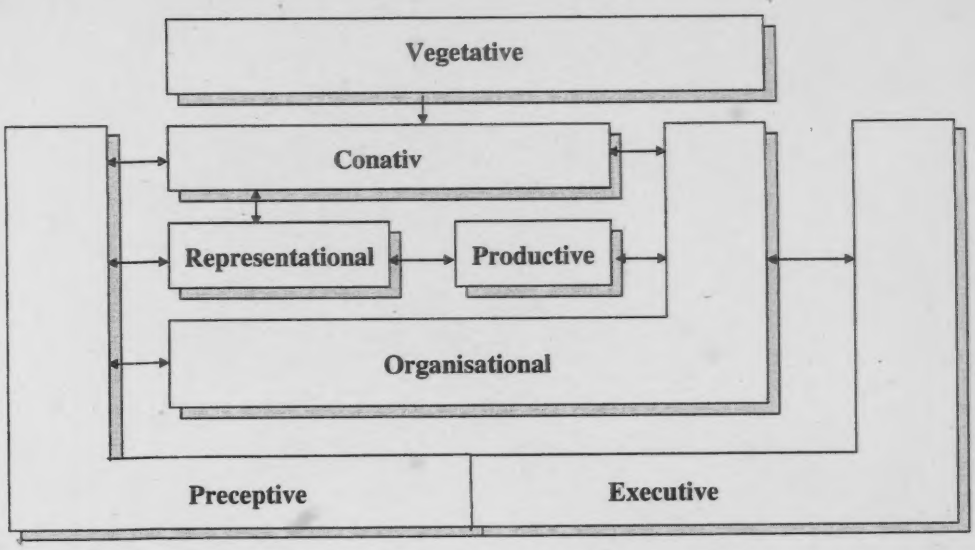


Figure 2.1: Representation of the Functional Components of an Autonomous Agent [34].

(and hopes, fears, intentions, hunches,...)" [27]. The use of intentionality is evidently useful for modeling systems externally but can also be used as a design tool. Assigning intentionality to a system allows the designer to capture that system's desired behaviour before assembling it.

The BDI approach has been used in many autonomous multi-agent systems [18, 56, 22]. The use of BDI has also been intimated in describing many other systems but not explicitly illustrated as a design process; this shows that BDI is a natural and logical approach to building systems of autonomous agents.

Many different approaches exist for ascribing intentionality to an autonomous system. The RoboLog Koblenz [84] RoboCup simulation team in 1999 represented beliefs as qualitative predicates and desires as goals or actions which must be satisfied by the actors or agents. Predicate logic is used to allow for deeper reasoning.

In the AT Humboldt [18] RoboCup simulation team, desires and intentions were decoupled to allow greater flexibility. In this system, the belief component was a representation of the agent's belief about the environment, the desire component would evaluate the expected success of potential desires and commit to a single one, finally the intention components would specify the best plan to achieve a committed desire. A similar approach was demonstrated by team Erika [56], also a RoboCup simulation league competitor in 1999, that represented desires as goal predicates similar to rules in Prolog which were to be satisfied by the agent's beliefs. Intention was represented as a planning module which would generate a sequence of actions that would, hopefully, lead to satisfying the goals or desires.

The examples of intentional systems given above consist of systems with the capacity to maintain an internal representation of the environment, otherwise known as *hysteretic* systems (see section 2.3.2). Intentionality does not have to be limited to these deliberative systems. Reactive systems model goals or desires as reflexive actions based on the current state of the environment, therefore ascription of belief, desire and intention is also possible in purely reflexive or *tropistic* systems. A tropistic system's beliefs will consist of the currently perceived state of the environment, no reasoning on future states can be performed.

Pure reactive agents will not possess any kind of notion of long term goals, their behaviour is focused on the immediate situation. Some reactive control systems include the motor schema-based methodology proposed by Arkin [4] and the subsumption architecture proposed by Brooks [15].

2.3.2 Tropistic and Hysteretic Agents

Agents are entities capable of keeping an account of their surrounding environment through perceptions, and change its state through actions. An agent's perception is its ability to classify and distinguish the different states of the environment in relation to its features as well as in relation to its actions. Perception can be considered as a function that relates values referred to as *percepts* or *stimuli* to the set of world states. If P_a is a set of percepts associated with a specific agent a, and Σ is the set of world states, then an agent's perception function can be defined as follows [34]:

$$\text{percept}_a : \Sigma \longrightarrow P_a$$

The function percept_a will associate a percept or stimuli to each world state. An agent's capacity to perform actions consists in producing an influence that will change the state of the environment. This is represented by the function *exec* defined as [34]:

$$\text{exec} : \text{Op} \times \Sigma \longrightarrow \Gamma$$

where Op and Γ describe the set of enactable actions and influences produced by the actions respectively. What happens between the input and output (i.e. between perception and execution) is the deliberation of the next action to take. The deliberation of actions is what effectively determines an agent's overt behaviour. Reactive agents will take reflexive actions based on the current state of the environment while deliberative agents will use more complex planning algorithms to deliberate the next action. Two classes of reactive agents will be considered below: *tropistic* and *hysteretic*.

Tropistic Agents

Tropistic agents are agents that act in a completely reflexive way to the state of the environment. These agents do not have goals or internal representation of the world; therefore execution is directly linked to perception. Mikhail Prokopenko et. al [71] describe tropistic agents using clockwork agents formally described as an n-tuple A_t (detailing the agent's perceptions, actions, the set possible world states and the functions that map perceptions to actions):

$$(P, \Sigma, \Gamma, \text{timer}, \text{command}, \text{response})$$

where P, Σ and Γ are defined as above and the functions *timer, command* and *response* are defined as:

$$\text{timer} : P \longrightarrow \Sigma \tag{2.1}$$

$$\text{command} : \Sigma \longrightarrow \Gamma \tag{2.2}$$

$$\text{response} : \Gamma \longrightarrow P \tag{2.3}$$

The *timer* function 2.1 describes the the sensory function of the clockwork agent. Whenever a clock event occurs the current state is updated. The *command* function 2.2 does a direct mapping from the current state to an action. Finally, the *response* function 2.3 provides the mapping from actions to an influence on the environment. In this case, a clock event is generated. Tropistic agents, as defined by Prokopenko et. al [71] are slightly different than the clockwork agent as the functions 2.1 and 2.2 are replaced by a single mapping from perception to action:

$$\text{sense} : P \longrightarrow \Gamma \tag{2.4}$$

This will cause the overt behaviour of the agent to be reactive to the current state of its environment. The n-tuple for a tropistic agent can be defined as follows:

$$(P, \Sigma, \Gamma, \text{sense}, \text{response})$$

It may be thought, a priori, that due to their simplicity, tropistic agents may not be very usefull. However, it should be possible for situated tropistic agents to exhibit relatively complex behaviour by using the environment itself as the model. The benefit of these types of agents is their adaptability to changing conditions in the environment and flexibility in accomplishing tasks [34].

Hysteretic Agents

Despite the interesting properties of tropistic agents (i.e. their ability to perform somewhat complex tasks with very simple agent design), it would be useful to have agents that can internalise information about experiences in the environment rather than simply using it as a model. The ability to preserve information requires agents to be endowed with some form of memory (other than environmental). These types of agents are called *hysteretic*. Prokopenko et al. describe hysteretic agents as "reactive agent[s] maintaining internal state I and using it as well as sensory states S in activating effectors E" [71]. For an agent to react, not only to its perceptions, but also to past experiences requires an internal state s_a belonging to the set of internal states S_a for an agent a. Memorising information or experiences therefore comes down to going from one internal state to another. Therefore the deliberation function can no longer be described as a simple reflex to precepts. Deliberation in hysteretic agents will take the form of two functions, a memorisation function *mem* and a decision function *decision*. These functions are described as follows:

$$\text{mem}_a : P_a \times S_a \longrightarrow S_a \tag{2.5}$$

$$\text{decision}_a : P_a \times S_a \longrightarrow \text{Op}_a \tag{2.6}$$

where Op_a has the same definition as above. A new perception function for the hysteretic agents, similarly to tropistic agents, can be defined as a function of the current world state. Optionally the internal state of the agent can also be considered for perception. In this case the *percept* function takes two arguments, the world state and the internal state:

$$\text{percept}_a : \Sigma \times S_a \longrightarrow P_a \tag{2.7}$$

Using the mem_a function 2.5 and the decision_a function 2.6 for the hysteretic agents, its response_a function can be defined. The response function for the hysteretic agent will be significantly more complex than that of tropistic agents as it must associate the pair (Σ, S_a) to the tuple (Op_a, S_a):

$$\text{response}_a : \Sigma \times S_a \longrightarrow \Gamma \times S_a \tag{2.8}$$

$$\text{response}_a(\sigma, s) = (\text{decision}_a(p, s), \text{mem}_a(p, s))$$

where

$$p = \text{percept}_a(\sigma)$$

The hysteretic agent class extends the tropistic class by adding an internal state, a hysteretic decision function and a function to update the internal memory structure. The hysteretic agent class can be described using the following n-tuple:

$$(P_a, \Sigma, \Gamma_a, S_a, \text{sense}, \text{decision}_a, \text{mem}_a, \text{response}_a)$$

Many problems are faced by designers of multi-robot systems, but there exists many tools and formalisms to assist the designer in their task. It is often useful, when designing robots intended to work as a team, to be able to look at the collective in an abstract way. This can allow designers to observe organisational patterns that may facilitate cooperation and communication within the collective. The natural world can also provided inspiration for useful abstractions for this purpose.

In order to understand how a collective of robots will cooperate to achieve joint-goals, it is also important to go beyond a global view of the collective and understand how each agent will behave individually. The individual behaviour of each robot in a collective, whether it is deliberative or reactive, tropistic or hysteretic, will affect the composition of the organisational structures withing a collective. Both of these views of the system shoiuld be considered to effectively construct and deploy a collective of cooperating robots.

Chapter 3

Telerobotics, Teleautonomy and Supervisory Control

Telerobotics and supervisory control are often considered to be the same thing; however there exist subtle differences between these two concepts. In fact, supervisory control can be considered as telerobotics but the opposite is not necessarily true; telerobotics is a proper super-class of supervisory control. In many cases, telerobots use closed loop control with the operator closing the loop. These standard telerobotic systems are predicated on a *master-slave* model where an *active* agent controls a *passive* entity. In this scenario, which occupies one extreme of a spectrum of the interaction models illustrated in figure 3.1, the passive entity lacks agency and thus has no innate goals, desires or intentions. A classic example of this *master-slave* model of interaction is the puppeteer and the puppet. The puppet cannot perform on its own and it relies on the skill of the puppeteer to put on a good show. Similarly, fundamental telerobots have no sensors providing feedback to the operator and are unable to issue commands to themselves, and therefore rely on the operator to direct them at the actuator level. They also rely on the faculties of the operator to analyse their environment.

At the other extreme, *active-active* agency implies that interaction is taking place between intentional agents (i.e. endued with beliefs, desires and intentions in their truest sense). This model embraces the idea that all points of the system are actively involved and

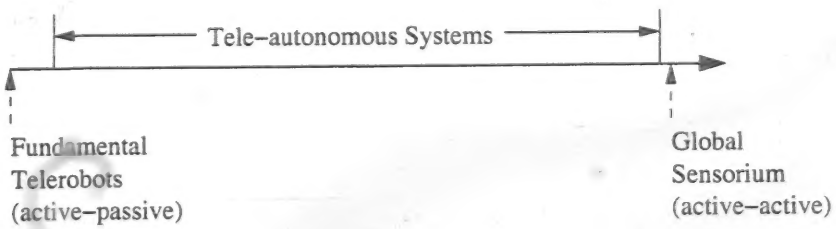


Figure 3.1: Spectrum of telerobotic interaction models.

their "perceptions and cognitions are melded together to form a global sensorium that might be likened to a neural network"[9]. Between these two extremes lies the class of telerobotic systems where both entities are active insofar as they possess some degree of autonomy, but also exhibit the qualities of a master-slave relationship. This class of telerobotic systems can be referred to as tele-autonomous. Going back to the example of the puppeteer and the puppet, in the fairy tale Pinocchio, Mastro Gepeto's wooden creation becomes sentient and the relationship between them suddenly shifts from *passive-active* interaction towards *active-active* interaction. However the elements of *master-slave* relation ship still endure as Gepeto is seen as somewhat of a father figure or a figure of authority.

Much work has emerged recently in the field of tele-autonomous systems [14, 25, 40]. An important distinction between this and standard telerobotics stems from the fact that the role of the operator is not vital to the overall functioning of the system. In this case the telerobots behave autonomously or semi-autonomously to achieve a task while being subject to an operator's influence. Conway, Volz and Walker defined teleautonomous technology as "The interaction of humans with remote, intelligent, partly autonomous systems of many forms" [25]. These systems merge autonomy with telerobotics to achieve a level of control that allows the operator to ignore certain details about the task being accomplished. The level of operator control required in these systems depends on the robot's abilities to sense its environment and deliberate a useful course of action.

Thomas Sheridan [81] provides a more pragmatic characterization of telerobotics and supervisory control. He gives a very intuitive definition of the basic supervisory control paradigm as a human operator providing *symbolic* or *analogic* commands to a computer. The difference between symbolic and analogic communication is as simple as the difference between communication through language and through gestures.

Language, spoken or written, is symbolic. It is formed of the elements of grammar along with rules governing their use. Words are the symbols of language and represent real life objects both tangible and intangible. Thus speech and the written word are two forms of symbolic communication. In terms of human-machine interaction, a symbolic command is one that is made up of a stream of symbols that are understood by both the operator and the computer. An example of the use of symbolic commands is the command line in the UNIX operating system; the human interacts with the computer using predefined shell commands (or symbols) that the computer understands.

Opposite to spoken or written language, body language is analogic in that it relates to experiences that are assumed to be shared between the communicating agents. Gestures can be used to communicate affection (blowing kisses) or rage (punching one's fist in the air). In terms of human-machine interaction, analogic commands take the form of physical control movements which are isomorphous to the space-time-force continuum of the physical

tasks. As an example, radio controlled cars use analogic commands – the movements of the control stick – to direct them. Analogic commands are useful as a means of demonstrating temporal-physical relationships to the computer that would otherwise be difficult to describe with symbols.

3.1 Some Definitions

Before continuing, it is important to define some concepts that will be used throughout this dissertation. The following terms are vital to understanding the subtle concepts of supervisory control and tele-autonomous systems:

Automation "Automatic, as opposed to human, operation or control of a process, equipment or a system; or the techniques and equipment used to achieve this. Most often applied to computer (or at least electronic) control of a manufacturing process."[1]

Robot A robot can be defined as a re-programmable mechanical device that can perform functions normally ascribed to humans. The term *robot* comes from the Czech word *robota*, which means work or compulsory service. Consequently the relationship between humans and robots is described as a master-slave interaction. Many believe that robots tend to look like humans and thus consider devices that do not fit this description nothing more than simple machines. This is false however as even a simple card shuffler is a robot as it performs the task of shuffling cards which otherwise would be done by the players. Modern science fiction has confused the definition of a robot with that of the thinking machine considered by Turing [91].

Telematics The term telematics refers to "the conjunction of computers and telecommunications" [79]. More specifically, telematics encompasses all computer-mediated communications, such as the Internet, and within the scope of this document, is considered as the medium through which humans and machines at different locations are able to cooperate. Telerobotics is a specialized sub-division of telematics.

Tele-effector The term tele-effector is used to describe a machine that extends a person's sensing and/or manipulating capability. The tele-effector will typically be located at a site remote from the agent who is operating it. Sheridan uses the term *teleoperator* to designate such a device however this term is confusing as an operator is properly defined as "an agent that operates some apparatus or machine"[2] rather than something that brings about a result or event. The term tele-effector is more suited to this definition.

[1] Source: The Free On-line Dictionary of Computing, © 1993-2001 Denis Howe.
[2] Source: WordNet 1.7 Copyright © 2001 by Princeton University. All rights reseved.

Telerobot A *telerobot* is an advanced form of a tele-effector whose behaviour is supervised by a human operator. "[T]he operator intermittently communicates to a computer information about goals, constraints, plans, contingencies, assumptions, suggestions, and orders relative to a remote task, getting back integrated information about accomplishments, difficulties, and concerns and (as requested) raw sensory data"[81].

Human operator, operator The operator is the active entity in a telerobotic system that monitors and controls its operation, using either direct or supervisory control .

3.2 Classifications of Telerobotic Systems

Human-machine interaction can be described using the theory and models of supervisory control. At the beginning of this chapter, a spectrum of interaction models was described and illustrated in Figure 3.1. In addition to this spectrum of agency, human-machine interactions can be classified according to the level of control an agent can have over another; be they intentional or not. Generally, these interactions can be classified into six main types: direct control, traded control, shared control, supervisory control, learning control and guidance by local experts [25].

1) Direct continuous control Direct control is characterised by the non-existence of an on-board controller and usually takes the form of direct position servoing. The robot is directed solely by the commands of the operator and has no ability to take any self-directed action. This is clearly the simplest form of teleoperation as the robot does not need to perform any complex sensor analysis and deliberation. It simply relays sensor data to the operator who will directly control its end effectors. The urban search and rescue robots from the University of South Florida [59] used direct control for navigating the robots while the on-board intelligence was responsible solely for the task of victim detection.

2) Shared continuous control The shared control paradigm is characterised by an integration of controller and operator signals. "The control tasks are not switched in a *bang-bang* control fashion. Rather, these tasks are smoothly integrated, and the level of control shifts gradually from the operator to the machine, and vice versa"[14]. An example of shared control interaction would be a guided mobile robot which has autonomous collision avoidance behaviour which will modify the paths selected by the operator. In this case, the operator input can be considered as part of the autonomous controller. If the robot controller is based on the subsumption architecture [15], then the operator input would be represented as a module in the behaviour network. Alternatively, if the controller is built using a schema based approach [4], the input signal from the operator would be summed

with that of the various inputs of the controller (weighted using a gain function) to arrive at the final output signal.

3) Traded control The traded control paradigm is characterized by a control scheme between the operator and the controller that is mutually exclusive. This means that at any given time during the operation of the tele-robot, either the robot controller or the operator, are directing it. The robot will never be directed by both of these entities simultaneously. This type of control allows the robots to perform their tasks autonomously while providing a mechanism for the operator to "wrest" control from the robot to perform tasks that were unexpected at controller design time or that may be giving the robot some amount of difficulty. This type of human-machine interaction is a little more complicated than the direct control paradigm because the robot must be able to operate autonomously as well as under the direction of an operator. An example of a traded control are automated flight systems that are installed in many aircraft. The autopilot system can be given control of the aircraft allowing the pilot a break from continuous hand flying, providing time to handle other cockpit duties. "Current aircraft use low-level controllers to handle ordinary situations, but rely on human pilot commands to respond when emergencies or unexpected situations (e.g. course changes) arise"[10].

4) Supervisory Control The supervisory control type of interaction is very similar to the shared control and traded control schemes. It is characterised by an autonomous telerobot which will receive instructions or "coaching" from the operator. The characteristics of supervisory control overlap to some extent those of the shared and traded control paradigms. For example, the telerobot is programmed to achieve its given task autonomously while the operator will supervise its progress and propose corrective measures, similarly to shared control. Alternatively, the operator can take control and complete the task as with traded control. In both cases, either the operator or the system can decide in which situation one of these interventions is necessary. Supervisory control is the most complex of the interaction types due to its polymorphic nature; it must be able to do the job of both traded and shared controllers.

5) Learning Control In this mode of operation, the telerobot or tele-effector is given some degree of intelligence to allow it to learn from the operator's input. The *slave* can subsequently deduce correct behaviour in similar situations without human intervention.

6) Guidance by local experts This final class of interaction is oriented more towards collaborative environments rather than telerobotics. "In this mode a variety of media, such as visual displays, graphics, touching, and pointing, are used to achieve a collaboration

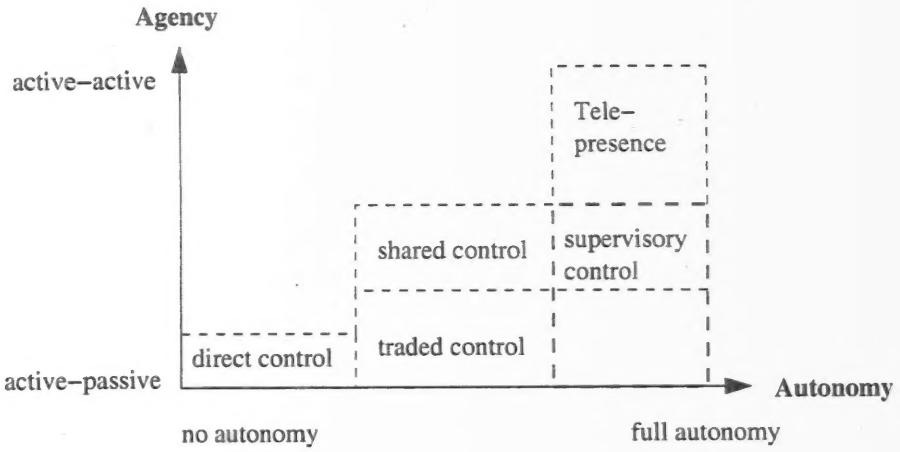


Figure 3.2: Matrix of agency vs. autonomy in telerobotic systems

between the local expert and the remote non-expert"[25].

Graves and Czarnecki proposed design patterns in [38] to represent four of the six interaction types. Figure 3.1 can be extended to represent the classes of human-machine interaction in telerobotic systems as a function of agency vs. autonomy. This is illustrated in figure 3.2 where the four classes of human-machine interaction listed above are plotted along with telepresence which is another technology that extends from telematics.

3.3 Collaborative Control

In Sheridan's definition of supervisory control [81], he does not preclude the possibility of having many supervisors for a single system. This type of interaction would allow telerobotic systems to break free from the typical *master-slave* relationship to one that involves *active-active* interactions. This is significant for modern expectations of the relationship between humans and machines. Telematics, it was recognised early on, could be used to "reinforce mechanisms of rigidity, authority, and domination" [61] or to build a "system of connections that will allow information and social organization to progress together"[61].

Collaborative supervision of a system of multi-robots could be useful in cases where the collective has to accomplish many different kinds of specialized tasks. Having many expert supervisors in this case would be more favorable than having a single supervisor who is proficient at most things. The challenge in the former lies in how to allow the multiple supervisors to work together without getting in each other's way. It is not difficult to imagine a scenario where a team of simple tele-autonomous robots (i.e. with one operator for one robot) could benefit from a supervisor with a global view of the work environment. The local and global supervisor would share control of the tele-autonomous device to accomplish

a joint task. A rather pedagogical example of this type of interaction is seen in Norman White's and Doug Back's telephonic arm wrestling system [94] which allowed a person to arm wrestle an opponent at a remote location. In this example, the agents at either end of the communication channel have conflicting goals, each wanting to win the arm wrestling match, yet each agent participates actively in the outcome of the match.

Some research has already been done in the area of collaborative agent technology. Hans Chalupsky et al. have developed "technology and tools for deploying agents into human organizations to help with organizational tasks"[20]. Their Electric Elves software agents were developed to help in collaborative tasks such as scheduling meetings and appointments etc. Each agent in the system has some degree of autonomy allowing it to make certain decisions on behalf of its human counterpart. Each agent is capable of adjusting its level of autonomy to maintain, as much as possible, the work flow of the system even if the operator is unable to explicitly provide input on decisions. This adjustable autonomy implies a dynamic interface (i.e. an interface that is situation dependent) between the user and the system. Applying such technologies to enable collaborative telerobotics could lead to very interesting and novel applications of tele-autonomous systems. When the day comes that AI programs are capable of passing the Turing test, human-system interaction will look more and more like human-human interaction and the social organization of these autonomous agents becomes that much more important.

3.4 Human Operator Control Interface

One of the major challenges that is faced during the conception of a human-system interface for a tele-autonomous robotic system is to satisfy the goal of simplifying, as much as possible, the user's task. This is the essence of supervisory control: to allow for the elimination of human action when it is either not needed or its result can be more efficiently achieved using automated action. Sheridan models this phenomenon as in the human relationship between a supervisor and subordinates.

> The term [supervisory control] is derived from the close analogy between the supervisor's interaction with subordinate human staff members in a human organization and a person's interaction with "intelligent" automated subsystems. A supervisor of humans gives directives that are understood and translated into detailed actions by staff subordinates. In turn, subordinates collect detailed information about results and present it in summary form to the supervisor, who must then infer the state of the system and make decisions for further action[81].

Humans Generally Better	Machines Generally Better
- Sense low level stimuli	- Sense stimuli outside human's range
- Detect Stimuli in noisy background	- Count or measure physical quantities
- Recognize constant patterns in varying situations	- Store quantities of coded information accurately
- Sense unusual and unexpected events	- Monitor prespecified events, especially infrequent ones
- Remember principles and strategies	- Make rapid and consistent responses to input signals
- Retrieve pertinent details without a priori connection	- Recall quantities of detailed information accurately.
- Select alternatives if original approach fails	- Process quantitative data in prespecified ways
- Reason inductively: generalize from observations	- Reason deductively: infer from a general principle.
- Act in unanticipated emergencies and novel situations.	- Perform repetitive preprogrammed actions reliably.
- Apply principles to solve varied problems.	- Exert great, highly-controlled physical force.
- Make subjective evaluations	- Perform several activities simultaneously.
- Develop new solutions	- Maintain operations under heavy information load.
- Adapt physical response to changes	- Maintain performance over extended periods of time.

Table 3.1: Relative capabilities of humans and machines. *Source* [82].

It is important to balance automation and human control in order to allow the system to be clear, flexible and capable of adapting to the external goals of the operator. One goal of introducing a human operator to a closed loop system is to remove complex cognitive tasks from the autonomous system that are much more easily achieved using the human reasoning system. The operator should concentrate on critical decisions, planning and coping with unexpected events and situations while the autonomous system can handle routine, tedious and error-prone tasks or tasks that require the analysis of large volumes of data that may either be too large to transmit to a remote operator site in a timely fashion or to voluminous to be easily interpreted or represented in an easily understandable abstraction. For these interpretative and representational tasks, automation is preferred since the potential for error can be reduced and, as software engineering methodologies become more standardized, hardware becomes more reliable, and software verification and validation improves, its use will become more prevalent. For many applications however, human operators will continue to have a critical role as the real world has a non-denumerable number of unpredictable events. Table 3.1 outlines the relative capabilities of humans and machines.

3.4.1 Theories of Interaction

A useful methodology for designing a useable interface prescribes the establishment of theories on how users will interact with the system. These theories can be formulated by observing and describing the stages of actions that users may go through when using a particular system. Donald A. Norman [62] proposes the following *seven stages of action* as a model of human-computer interaction:

1. Formation of a goal.
2. Formation of an intention.
3. Specification of an action.
4. Execution of the action.
5. Perceiving the system state.
6. Interpreting the system state.
7. Evaluating the outcome.

The seven stages of actions provide, for the human-system interface, the same level of abstraction as the practice of *separation of concerns* in software engineering. Separation of concerns refers to the ability to identify, encapsulate, and manipulate parts of a software system that are relevant to concepts, goals, tasks and purposes [68]. Theories of interaction will therefore be fleshed out by analysing the way people perform their jobs. This process is referred to as *task analysis* [28]. Task analysis techniques will usually involve the decomposition to express the behaviour of a global task. This decomposition will result in a hierarchy of tasks, subtasks and plans describing the order in which, and conditions for which these sub-actions should be performed.

Another approach to task analysis, knowledge-based analysis, involves the creation of taxonomies of objects and actions involved in a particular task. "The aim is to understand the knowledge needed to perform a task and thus to help in the production of teaching material and in assessing the amount of common knowledge between different tasks"[28].

The uses for task-analysis are as follows: requirements capture, system design and interface design. Given a hierarchy of objects and tasks, it becomes easier to establish the requirements of the user-interface and thus its design is also simplified. The design of a new, but not necessarily novel, system can be guided using the analysis techniques described above. Designers can, to the extent of their understanding of how a system should work, determine what objects and tasks require user input and those that should be automated. For a new system, this task analysis can help in structuring the internal model for

the system that matches the user's expectations. From the perspective of interface design, the taxonomies obtained during the analysis phase can be used to, for example, determine the layout of menus etc.

Telerobotics, teleautonomy or supervisory control allows human input to be provided to a robotic system at a remote work site. The utility of doing this can be understood by looking at the relative strengths and weaknesses of humans and machines. The dynamics of the system depend on the relationship between the human operator and the robotic effector located at the work site. This robotic effector may or may not be autonomous, and the level of control over the effector available to the human may also vary. Based on these factors, telerobotic systems are classified into six general categories. The selection of a class of telerobotic control for a particular application should simplify the operators task as much as possible.

Chapter 4

The SimCanuck Tele-Autonomous Soccer System

This chapter will describe the totality of the SimCanuck tele-autonomous soccer system designed for use with the RoboCup soccer simulation server. The system is composed of two major parts: a system of semi-autonomous soccer agents and an interface used by the human operator to guide it. Each component of the system will be described in detail in the following two sections.

4.1 Semi-Autonomous Soccer Agents

The simulated robots, or agents, used for the tele-autonomous soccer system are considered as being semi-autonomous. They act in a mostly autonomous way but will also make use of advice provided to them by a human operator. In actual fact, these robots were designed to be able to work completely autonomously (i.e. without human intervention); therefore the term semi-autonomous seems a little restrictive as a description of them. The main elements that each robot lacks, which are provided by the coach, are plans, goals and intentions. Without the human operator, the robots are nothing more than strictly tropistic agents (i.e. purely reactive).

Each robot has three major components: a memory, a playbook and a behaviour control unit. The memory component represents the robot's current belief of its work environment. Visual and auditory information received from the soccer server is stored in this memory structure and can be queried by the robot to obtain information about the current state of the environment.

The playbook component is where information on plans, missions and goals will be stored. If the playbook is not used, the robot will simply react to the state of the environment represented in the memory component. Information received from an on-line coach client will be interpreted and stored by this component. An artificial coach may also be used with the SimCanuck robots; however the playbook component was designed to facilitate

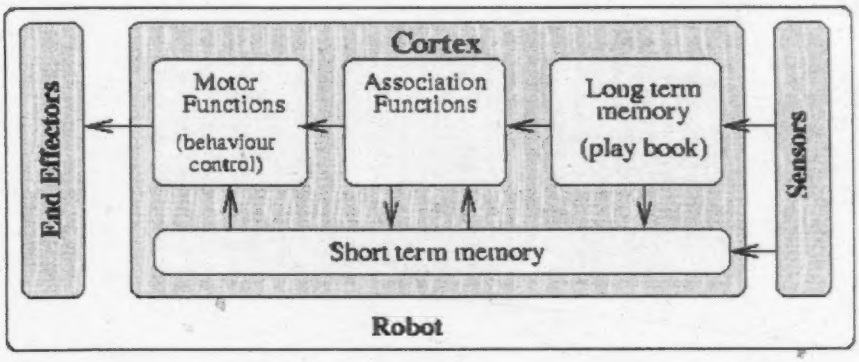


Figure 4.1: A diagrammatic view of the components of the SimCanuck robots organised into a "cortex" like structure.

interaction with a human not a machine. The interpretations of coach advice may not satisfy the constraints of an artificial coach client for them to be of any use to the robots.

The final component is the behaviour control unit. This consists of a hierarchy of low level behaviour modules organised in a subsumption architecture. The behaviour modules used by the SimCanuck robots are of a higher level of complexity than those considered by Rodney Brooks when he originally proposed the subsumption architecture [15]. Instead of relating directly to motor control commands, each module encapsulates a behaviour specific to the robotic soccer domain. For example, the action of kicking the ball towards the goal is encapsulated in a single behaviour module. The implementation of each one of the individual behaviours may themselves use a subsumption architecture. However the goal of representing more complex actions as single behaviour modules allows the very low level implementation details of the system to be abstracted. By drilling down to lower levels of abstraction (i.e. down to the motor control level), a more elaborate behaviour hierarchy may be exposed.

These three components put together can be seen as fulfilling the role that the cortex does in a human or animal brain. It provides the motor and associative skills necessary to enable the robot to act "intelligently" in its environment. The SimCanuck "Cortex" is illustrated in Figure 4.1 where the *motor functions* component represents the behaviour controller and the *association functions* component enable the integration of playbook information, stored in the *long term memory*, with the world representation in the *short term memory*.

4.1.1 Behaviour Controller

The most important component of the cortex described above is the motor controller unit. This component prescribes how each robot will interact with their environment and is the source of their overt behaviour. As stated above, the behaviour controller is implemented

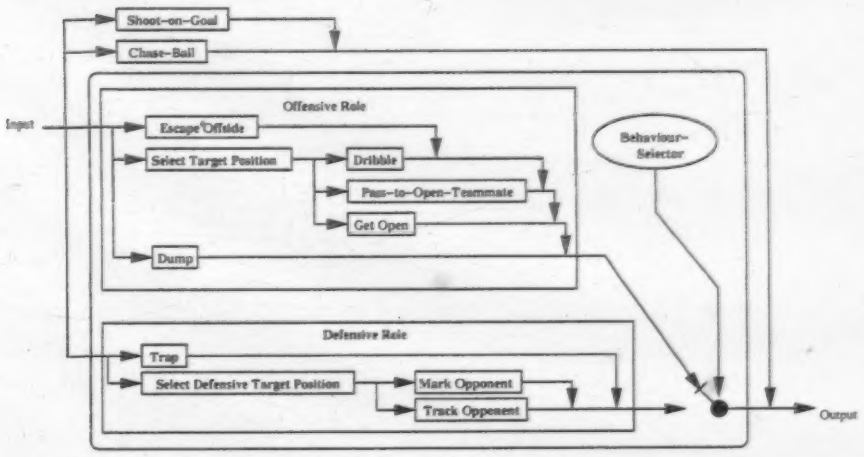


Figure 4.2: A diagrammatic representation of the subsumption architecture of the behaviour controllers for robots fulfilling the forward, defender or mid-fielder roles.

as a hierarchy of behaviour modules organised as a subsumption architecture. Each robot, with the exception of the one fulfilling the role of the goalie, will have an identical behaviour hierarchy with every other robot in the collective. The goalie's controller will be quite different due to the specialised nature of its role. The goalie agent is less constrained than any other robot on the play area or pitch; the goalie is the only robot capable of picking up the ball. This additional skill available to the goalie robot makes it desireable to have it protect the goal area. Since the ball can only be picked up when it is in the penalty area, it would be unwise for the goalie to leave this part of the field. Each of the controllers and their respective behaviour modules will be described below.

4.1.1.1 Behaviour Hierarchy for the Forwards, Mid-Fielders and Defenders

A diagram of the behaviour hierarchy for the robots fulfilling the roles of forwards, defenders and mid-fielders is illustrated in Figure 4.2. The controller consists of a total of twelve behaviour modules and a single behaviour selector. Some of the modules that are closely related are grouped together and are intended to be active only if the behaviours of another grouping are not. The controller diagram in Figure 4.2 has two such groups: the offensive role and the defensive role. A behaviour selector is a specialised module which will select an appropriate grouping of behaviours which should be active in the current state of the environment. In Figure 4.2, the output of the offensive or defensive behaviours will be selected according to the agent's belief about the possession of the ball. In other words, if the agent believes that its team is in control of the ball, the offensive behaviour output will be selected; otherwise the defensive behaviour output will be chosen.

The behaviour selector could also be represented as another module in the behaviour

hierarchy with a single output line for each behaviour in each grouping under its control. The selector behaviour would then simply route the input signal only to those behaviours that are in the currently active grouping.

The outputs of each behaviour modules consists primarily of commands to send to the soccer server (i.e. turn, dash, kick, turn_neck etc.). Besides the behaviour selector modules, the twelve modules in the controller for the forwards, mid-fielders and defenders are as follows:

Shoot On Goal This is the top level behaviour (i.e. it has the highest priority). The *shoot on goal* behaviour will be active if, and only if, the robot is in possession of the ball in a good position to take a shot on goal. The output of this module will therefore, naturally, be a kick command. The parameters of the kick command will be calculated by this module such that the shot will be taken towards an open section of the opponents goal (i.e. not where the goalie is standing).

Chase Ball This module has the second highest priority and will be active if the robot believes it is the closest of its teammates to the position of the ball. The output of this module will be either a *turn* or a *dash* command that will guide the robot towards an optimal position to take control of the ball; this may be either an estimated position or an absolute position. If the ball is believed to be in motion, then the *chase ball* module will estimate an interception point and the output of this module will guide the robot to that point. Otherwise the output will guide the robot towards the ball's absolute position.

The next modules are part of the groupings of behaviours used when fulfilling an offensive role and defensive role. The behaviours represented by these modules will be either active or inactive as a group depending on the role the behaviour selector currently has selected. This means that if any of the offensive role behaviours are active, then *none* of the modules in the defensive role must be active. Each grouping must therefore be mutually exclusive.

Offensive Role Behaviours

The offensive role grouping encapsulates all the behaviours that are relevant to offensive play. These include behaviours for dribbling, passing and choosing a position in which to receive a pass. The shoot on goal behaviour is not part of this grouping because if there is the possibility for a shot on goal, even though the currently selected role is not the offensive one, the shot should be taken.

Escape Offside This is the highest priority module in the offensive role behaviour grouping. The escape offside behaviour will be active if, the robot is not in possession of the ball, and it believes that it is currently in an offside position. The output of this

module will be either a *turn* or a *dash* command that will guide the robot towards the nearest on-side position.

Select Target Position This module will select an x-y coordinate that represents a suggested offensive position that the robot should consider when evaluating its actual desired position. This behaviour will serve as the input for the *Dribble, Pass To Open Teammate* and *Get Open* modules. The output of this behaviour is an x-y coordinate that serves as a suggested target for modules which use this one as an input.

Dribble This module will be active only if the robot is currently in possession of the ball. The *Dribble* behaviour takes its input from the *Select Target Position* behaviour. The input is used in evaluating a target to which to dribble. The output of this module will be a command, either *kick, turn* or *dash* which will direct an evaluated target that is based on the input target. If the robot determines that the dribbling action is impossible, then this module will not produce any output.

Pass To Open Teammate This module will be active if the robot is in possession of the ball and there is an open teammate to pass to. This behaviour is directly subsumed by the *Dribble* module. If the robot is unable to dribble to a specified position, the overt behaviour will then fallback to passing the ball to a teammate who is close to the desired target. The *Pass To Open Teammate* behaviour takes its input from the *Select Target Position* module and finds an optimal candidate for a pass. If the module determines that a pass is not possible at this stage, the module will produce no output (and therefore not subsume lower priority modules).

Dump This module which will also be active if the robot is in possession of the ball will guide the robot to kick the ball up the field; away from its own goal and towards the opponent's goal if possible. This is the fallback behaviour if the robot can neither dribble nor pass. This module will always produce an output, and therefore subsume lower level outputs, on condition that the robot is in possession of the ball. Otherwise the *Dump* behaviour will not produce an output. The input target obtained from *Select Target Position* will be used as a point of reference for clearing the ball. The ball will therefore likely be "dumped" towards the target specified in the input of this module. The output of this module will consist of a *kick* command that will guide the robot to kick the ball towards the evaluated clearing target.

Get Open This module will be active if the robot is *not* in possession of the ball. This behaviour directs the robot to position itself in an open part of the field to receive a pass from a teammate. An input target from *Select Target Position* is used as a point of reference for the evaluation of a new position by this module. The output of this

behaviour will be either a *turn* or a *dash* command which will guide the robot towards the open position that was evaluated.

Defensive Role Behaviours

The defensive role encapsulates the behaviour modules that are relevant to defensive play. These include the *Trap, Mark Opponent* and *Track Opponent* behaviours. The *Chase Ball* behaviour is not included in the defensive role grouping as it should be possible to activate it regardless of the currently selected group of behaviours. Chasing the ball is useful in both offensive (dribbling) and defensive (intercepting passes) situations.

Trap This is the top level module (i.e. with the highest priority) in the defensive role grouping. This behaviour will be active if some action can be taken to put an opponent's player in an offside[1] position. The output of this module will be either a *turn* or a *dash* command which will position the robot such that the opposing team will end up in an offside position.

Select Defensive Target Position This behaviour fulfills the exact same role as the *Select Target Position* in the offensive role behaviour grouping. The *Select Defensive Target Position* module will select an x-y coordinate that represents a suggested defensive position that should be considered in the evaluation of the actual defensive position. This module provides the input for the *Mark Opponent* and *Track Opponent* behaviours.

Mark Opponent This module will choose an opponent player to mark and guide the robot towards it. The selected mark will be chosen based on the target supplied as input from the *Select Defensive Target Position* module. The output of this module will be either a *turn* or *dash* command which will guide the robot to an appropriate position near the selected opponent robot. From this position the robot should be capable of cutting the angle between the opponent and the goal being defended as well as intercept passes destined to the mark.

Track Opponent This module will select a position that the robot should occupy in order to cut the angle between an opponent and the goal. This behaviour is similar to the *Mark Opponent* module except that the robot does not need to be near the opponent being tracked. This behaviour will often times be used to set up a mark if for example the play is still on the opponent's side of the field. A defender will position itself such that when the action moves to its side of the field, it will be in a good position to

[1] A player is in an offside position if: he is nearer to his opponent's goal line than both the ball and the second last opponent.

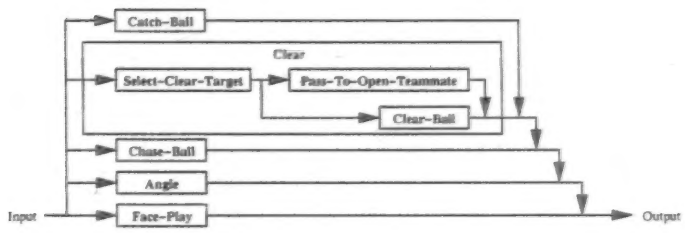


Figure 4.3: A diagrammatic representation of the subsumption architecture of the behaviour controller for the robot fulfilling the role of the goalie.

either angle the play away from the goal or to mark an opposing player. The output of this module will be either a *turn* or *dash* command.

4.1.1.2 Behaviour Hierarchy for the Goalie

A diagram for the controller of the robot that is fulfilling the role of the goalie is illustrated in figure 4.3. The goalie's controller is, by far, much simpler than for the forwards, defenders and mid-fielders. This is due to the fact that the duties of the goalie are much more restricted than those of other players (but possibly of greater importance). The goalie's controller consists of seven separate behaviours. As with the previous controller, some behaviours that are closely related are grouped together however the grouping is purely logical in this context. The goalie controller does not have a behaviour selector (i.e. there are no groups of modules who's activation will be mutually exclusive to another groups). The behaviour modules that make up the subsumption architecture for the goal keeper are described as follows:

Catch Ball This is the top-level behaviour (i.e. with the highest priority). This module will be active if, and only if, the robot is capable of catching the ball (i.e. the ball is within the catchable distance threshold). This is the most important behaviour as a goalie that cannot catch the ball will likely not be an effective goalie at all. The output of this behaviour module will be a *catch* command parameterized to guide the robot to attempt to catch the ball (this action is not guaranteed in the simulation). Note that the goalie robot is the only one who may issue a *catch* command to the soccer server.

Chase Ball The goalie's version of the chase ball behaviour is almost identical to that of the forwards, mid-fielders and defenders, the only difference being that the goalie will not chase the ball outside of the penalty area. The output of this module will be either a *dash* or *turn* command.

Angle This module guides the robot to position itself to cut the angle between an opponent

in possession of the ball and the goal being defended. The main goal of this behaviour is to reduce the amount of the goal that is visible to an opponent in shooting position. The output of this module will be either a *dash* or *turn* command. This behaviour will be inactive if the ball is not currently within the robots visual field (view cone).

Face Play This module will direct the robot to face the play (i.e. the position of the ball). It is of utmost importance for the goalie to be aware of the ball's position, especially when it is in the defensive half of the field. However, if the goalie spends all of its time fine tuning its position to have the optimal view of the ball, it may not be able to perform the necessary actions to position itself to block a shot. This is the motivation for giving such a low priority to this module. This is possible because none of the behaviours that subsume this one will ever be active if the ball is not within the current visual field of the robot. This fallback behaviour will therefore guide the robot to find the ball and centre it in its visual field. If the ball is in fact in the visual cone of the robot, it will direct the robot to adjust its position so that the ball will be centred. The output of this module will be a *turn* command, coupled with either a *turn_neck* or *change_view* command. The *change_view* is used to change the size of the view cone and the quality of visual information received. For the most part, the goalie robot will set its visual field to the largest possible size to see as much of the play as possible. The size of the visual field may be reduced the closer the ball comes to the goal being defended.

Clear Behaviours

The clear behaviours are grouped together logicaly. These modules are responsible for clearing the ball from the penalty area when the goalie takes possession of the ball (i.e. either by intercepting it or by catching it with a *catch* command).

Select Clear Target This module is similar to the *Select Target Position* behaviour in the controller for the forward, mid-field and defense robots. It will select an x-y coordinate that represents a suggested clearing target which should be considered when evaluating the actual clearing target (either a teammate or an arbitrary position away from the goal). This behaviour will provide the input for the *Pass To Open Teammate* and *Clear Ball* modules.

Pass To Open Teammate This module is exactly the same as *Pass To Open Teammate* module described previously.

Clear Ball This behaviour will direct the robot to clear the ball from the penalty area towards an arbitrary position away from the goal being defended. This is the fallback behaviour if the goalie cannot find a teammate that would be a suitable recipient for a

pass (i.e. closely marked by an opponent or out of range) and is active if, and only if, the goalie is in possession of the ball. This module will, as much as possible, clear the ball away from opponent players and closer to teammates. The output of this module consists of a *kick* command parameterised to perform the appropriate kick action.

4.1.2 Missions, Tasks, Plans and Actions

Planning is the act of finding task sequences that will achieve an agent's goals. This action is mostly applied to deliberative systems where goals or desires are modeled in the system. However, hybrid systems have also been developed which allow the explicit formulation of goals while using a reactive control scheme [7, 66].

The autonomous soccer agents used for this project have very little in terms of missions, plans and goals when they are first deployed. The reasoning behind this design decision is that they will be more flexible and thus more able to adapt to unexpected situations in the environment when provided with external advice. In order to benefit from these however, each robot must be able to understand the differences between *missions*, *plans* and *actions* and be able to perform a mapping from each of these entities to its own set of low level commands.

It is important to make the distinction between missions, plans and tasks. This distinction is useful for achieving cooperative goals; joint activities typically consist of many parts performed concurrently, conditionally or sequentially [24]. In the context of the current control architecture, a mission consists of a set of tasks that can be achieved by an ordered sequence of plans. It is defined in a very abstract way to describe some high-level goal.

A plan is represented as an ordered sequence of actions that, once executed, direct the robot to accomplish a given sub-task. A plan must be composed of at least one action which may be the *null* action (i.e. the robot does nothing). The null action allows a distinction between explicitly doing nothing and the absence of any plan or mission. This is important as the latter can be considered as an inconsistent and undesirable state. Let a plan ρ be defined such that:

$$\rho = \{\alpha_1, \alpha_2, \ldots \alpha_n\}$$

where $n \geq 0, \alpha_i \in \mathcal{A}$, and $\mathcal{A}$ is the set of valid actions for a particular robot in the current environment. Each action α_i in ρ may possibly represent a robot command (i.e. to guide its various effectors). For example, the set of commands available to simulated robotic soccer players using the RoboCup Soccer Simulator (RCSS)[2] includes (among others) *kick*, *dash*, *turn_neck* and *turn*. A plan ρ_{dribble} to dribble the ball to a certain point may be a sequence of actions similar to

$$\rho_{\text{dribble}} = \{\text{dash}_0, \text{kick}_1, \text{turn_neck}_2, \text{turn}_2, \text{dash}_3, \ldots\}$$

[2] RoboCup Soccer Simulation Server, http://sserver.sourceforge.net/home/index.html

The parameters for each of the actions in ρ_{dribble} are not shown but would be such that the action kick_1 would send the ball to within range of the following turn_2 and dash_3 commands. Similarly the dash_0 command would place the ball to within the robots kicking range (in order for it to be able to execute the specified kick).

The sequence of actions that make up the plan definition will typically require to be executed in the given order for the plan ρ to accomplished. It may be possible however for certain actions to be executed concurrently. In the example plan ρ_{dribble} given above, the turn_neck_2 and turn_2 commands may be executed simultaneously (as defined by the RCSS protocol). Each plan ρ_i therefore can be represented by a poset $(\rho_i, \preceq)$ where some elements may be uncomparable (i.e. the poset *may* be partially or totally ordered). The plan ρ_{dribble} is partially-ordered as the actions turn_neck_2 and turn_2 are uncomparable. In this example, the subscripts of the individual actions serve as the basis for ordering the actions in the poset. Actions with the same subscript may or may not be executed concurrently; therefore the plan ρ_{dribble} is equivalent to the plan

$$\rho'_{\text{dribble}} = \{\text{dash}_0, \text{kick}_1, \text{turn}_2, \text{turn_neck}_2, \text{dash}_3, \ldots\}$$

It may also be desireable to be able to generate higher level actions, or *hyper-actions*, by combining two or more basic actions that can be executed concurrently. For example, in the plan ρ_{dribble}, the first *turn* and *turn_neck* actions can be executed simultaneously, we denote a hyper-action combining these using the following notation:

$$\alpha = < \text{turn}_2, \text{turn_neck}_2 >$$

The plan ρ_{dribble} can therefore be re-written as

$$\rho_{\text{dribble}} = \{\text{dash}_0, \text{kick}_1, < \text{turn}_2, \text{turn_neck}_2 >, \text{dash}_3, \ldots\}$$

to explicitly express the concurrency of the *turn* and *turn_neck* actions.

In highly dynamic environments, plans may need to be modified in mid-execution in order for their specified end-results to be achieved. This is especially true in a situation where the number of actions n in a given plan ρ is fairly large. If $n \gg 1$, it is more likely, especially in an adversarial environment such as robotic soccer, that changes to the environment may interfere with the execution of a set plan. For this reason, the SimCanuck soccer robots will use open-ended plans where the end-result of the plan is considered but the sequence of actions is deliberated by the behaviour-based controller as needed. This approach to executing plans is illustrated as follows:

$$\rho_i = \{\ldots \alpha_{t-2}, \alpha_{t-1}, \alpha_t, \rho'_i, \alpha_f\} \tag{4.1}$$

The plan described by 4.1 is fully defined up to time step t; therefore the action α_t is the next action to be executed and the actions $\alpha_{t-1}, \alpha_{t-2}, \ldots$ have already been executed. An

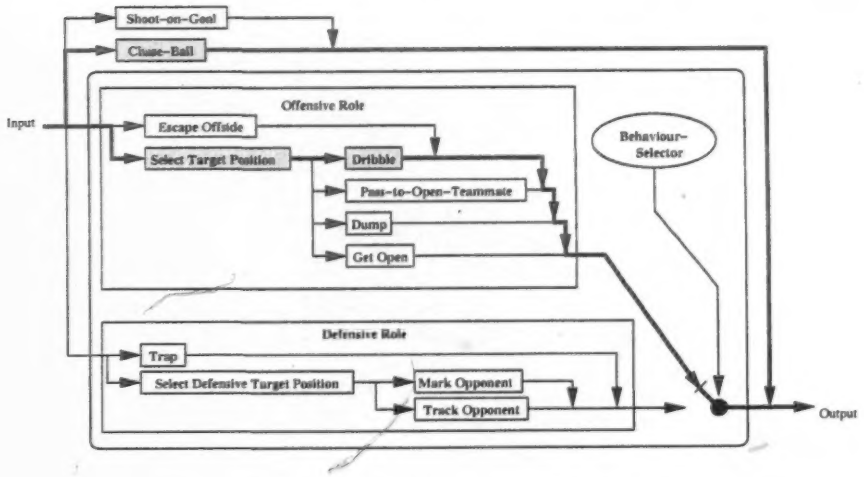


Figure 4.4: Active behaviour modules during the execution of a plan that involves dribbling the ball to a specified point (illustrated in gray) and the flow of signals between these modules (bold signal lines).

undefined plan ρ_i' indicates that the plan ρ_i has not yet been fully executed (i.e. the desired result has not yet been achieved). Therefore the robot will continue to try to execute ρ_i until it has either been successfully completed (the stub action α_f has been executed), or it is determined that the successful execution of the plan has been compromised by some environmental change rendering it impossible. This execution of partial plans is handled by specialised behaviour modules in the subsumption architecture used by the SimCanuck robots. For example, the "*dribble to point*" plan defined in ρ_{dribble} would be handled by a behaviour, or a network of behaviours, specifically designed for that purpose. In the case of the SimCanuck robots, the action of dribbling to a point is handled by three different behaviour modules: The *select_target* module is responsible for setting the target towards which the robot should dribble the ball, the *dribble_kick* behaviour uses the specified target and commands the robot to kick the ball toward it, and finally the *chase_ball* behaviour, which becomes active once the ball is kicked, commands the robot to dash towards the ball's position (estimated according to the time it will take the robot to get there). The network of behaviours used to handle the "*dribble to point*" plan is illustrated in Figure 4.4 within the context of the full controller diagram for the SimCanuck behaviour network.

The next level of abstraction above that of plans is the concept of a *task*. The definition of a task is composed of a set of plans (much like a plan is composed of a set of actions). Let a task τ be defined as follows:

$$\tau = \bigcup_{i=0}^{n} \{\rho_i\}$$

where $n \geq 0$. Only one task will be undertaken by any robot at any given time and

its successful completion will typically require that all the plans that make up the task definition be executed and accomplished. The scope of a task definition is restricted to the deliberation process of the current robot; the execution of the task can not be shared with other robots in the collective (i.e. a joint task). The execution of specific tasks may not be mutually exclusive however as another robot in the collective may possibly be executing the same task. Although more than one robot can execute the same task, in order for each to consider it as completed, they must execute it in its entirety. In some cases this may lead to a conflict if both robots require a common environmental resource.

The final level of abstraction is captured by the concept of missions. A mission is used to represent a joint task and will be composed of a set of tasks τ_i which will be distributed among the members of a robot or agent society. Let a mission μ be define as:

$$\mu = \{\tau_1, \tau_2, \ldots, \tau_n\}$$

The major difference between missions and tasks, besides the obvious level of abstraction of the information contained within, is that the scope of a mission may span many robots. A task definition on the other hand is only valid within the scope of the deliberative process of a single agent or robot. A set of tasks can be used to represent a joint goal or joint task as defined in [24] by Cohen *et al.* This difference is significant because the completion of a mission, from the perspective of a single robot in a collective, does not require it to execute and accomplish each of the tasks laid out. These tasks are typically meant to be distributed among the members of the collective in order to accomplish the joint task that is represented by the mission definition.

Missions, tasks and plans are very similar in the sense that, at their most basic level, they consist of an ordered sequence of tasks, plans and actions respectively. What this means for the interpretation of a mission is that, for the simplest case, a mission may be composed of a single task which itself may be composed of a single plan which may further be composed of a single action. This type of mission specification may have no significance for a system of multiple autonomous robots; however it clearly illustrates that missions may delineate goal-achieving functions of any complexity.

The principle motivation behind the decomposition of missions, tasks and plans discussed above is to facilitate their representation in each agent's memory. However this decomposition is also beneficial for the communication of mission information to the collective from an outside source. Mission specifications in the SimCanuck framework are based on the on-line coach language protocol used by the Soccerserver and is described in detail in Appendix B. The syntax of the coach language CLang is similar to that of the programming language LISP. The language allows regions to be defined and represented by string constants. These regions can then be used in *advice* messages to communicate mission intentions to the team of agents. The specification for a cross-play mission is illustrated in Figure 4.5 . This is a

```
1:  (define
2:    (definer "left_corner" (quad (pt 37.5 20.0) (pt 52.5 20.0)
3:                                 (pt 52.5 30.0) (pt 37.5 30.0)))
4:    (definer "opp_penalty" (quad (pt 35.0 15.0) (pt 45.0 15.0)
5:                                 (pt 45.0 -15.0) (pt 35.0 -15.0)))
6:    )
7:  (advice (3000 (bowner our {})
8:                (do our {} (bto "left_corner") {p d c})
9:                (do our {6 7 8 9} (pos "opp_penalty"))
10:               )
11:         )
12: (advice (3000 (and (bowner our {}) (bpos "left_corner"))
13:                 (do our {} (bto "opp_penalty" {p d}))
14:                 )
15:         )
```

Figure 4.5: Specification of a *cross play* mission using the on-line coach protocol defined for the RoboCup Simulation Server.

two step mission (i.e. it defines two missions which must be executed in sequence). The first guides the collective to move the ball, either by passing, dribbling or clearing it, towards the area to the left of the opponent's goal (defined by *left_corner* on line 2). The second task, which is addressed to specific robots (numbered 6, 7, 8 and 9), guides these robots to position themselves in the opponent's penalty area (defined by *opp_penalty* in line 4). Once both of these tasks have been executed and the goals they prescribe achieved, the first mission can be considered as completed and the second mission can then be undertaken.

The second mission specification has only one task, to move the ball into the penalty area, either by passing it to an open player that moved there as directed by the first mission, or by dribbling toward it. This will complete the cross play when the *shoot on goal* behaviour becomes active and the ball is shot on goal. This behaviour is placed into the hierarchy in such a way that it will have the highest priority of all modules. However it remains inactive until the robot is in possession of the ball in what is deemed to be a good scoring position.

The specification in figure 4.5 omits the details of what plans and actions are used to accomplish the various tasks specified. These details are left up to the SimCanuck agents who are in a better position to measure their own capabilities. The first mission specification on line seven of Figure 4.5 consists of a single joint-task of setting up the cross play. This joint task is divided into two different sub-tasks that must be distributed across the collective. The first task, defined on line 8, uses the *bto* (ball to) command which advises that the ball should be moved to the specified point; in this case to the *left_corner* as defined on line 2. This directive is addressed to whatever robot is in possession of the ball. Given this task, a robot in possession of the ball will either dribble it towards the corner or, if a teammate is closer to the specified corner, will try to pass it towards them. In the worst case, if neither of these actions are possible, the ball is simply cleared (kicked) toward the corner in the

hopes that something productive will happen.

The second task, defined on line 9, uses the *pos* command which advises the addressed players to move to the specified position; in this case to the *opp_penalty* area as defined on line 4. This directive is addressed to the robots numbered 6, 7, 8 and 9 explicitly. If any of these robots are in possession of the ball, this task will not be undertaken as this behaviour will be subsumed by a ball handling behaviour; the task described on line 8 will be undertaken instead.

In the case of the SimCanuck system, the order in which tasks are defined is important. A task that is defined earlier will have a higher priority than any task that is defined after it. This has the important side-effect that, when undertaking a specified mission, a robot will go through the various tasks and execute the first one that satisfies its current belief state. Other schemes for assigning priorities to task definitions could have been used, however this simple strategy can be shown to be sufficient.

The next mission specification on line 12 of Figure 4.5 defines the second part of the *cross-play* multi mission. This part of the multi mission will only be active if a teammate is in possession of the ball in the *left_corner* of the field. This is illustrated by the condition associated with the advice message on line 12 which consists of the conjunction of two conditions. Line 13 defines the task to undertake in order to accomplish the mission, it directs the team to move the ball towards the opponent's penalty area where one of the teammates numbered 6, 7, 8 or 9 should be located (assuming the task on line 9 has been undertaken).

4.1.3 Plan Generation

An important difference between tasks and plans is that plans are relatively stable over time. For instance, if a robot is assigned a specific task, the sequence of plans required to accomplish the task will differ based on the current state of the world. On the other hand, once a plan has been selected, the agent is committed to finishing the plan before computing a new one. In a reactive control scenario, plans are computed on an as needed basis. For the SimCanuck robots, a plan will generally be composed of one, two or three actions. For example, when dribbling the ball, a robot may want to dash towards the ball's current or estimated position and re-orient its visual cone to keep the ball in view. The current plan ρ_1 for achieving the dribbling task therefore will consist of two actions:

$$\rho_1 = \{\text{dash}, \text{turn_neck}\}$$

both of these actions will be executed in a single discrete simulation step. Similarly, the goalie robot may be trying to accomplish the task of positioning itself to minimize the amount of the goal visible to an opponent in possession of the ball. If the opponent is far away

this may require the robot to, in addition to positioning itself, change the characteristics of its view cone. The plan ρ_2 to accomplish this will therefore be composed of three actions:

$$\rho_2 = \{\text{dash}, \text{turn_neck}, \text{change_view}\}$$

Hysteretic agents will build a set of ad hoc plans to accomplish a task where each successive plan is deliberated just in time. In the robotic soccer simulation environment, time changes are modeled in discrete intervals. The generation of these ad hoc plans can therefore be represented as a mapping from the world state and current task to a plan that accomplishes it.

$$\text{plan} : \Sigma \times \mathcal{T} \longrightarrow \mathcal{P}$$

where Σ is the set of all possible world states, $\mathcal{T}$ is the set of all possible tasks (possibly infinite) and P is the set of all plans which may be used in accomplishing a task $\tau \in \mathcal{T}$.

Given an instance of a task $\tau \in \mathcal{T}$, it can be assumed that the plan generated by the above function is incomplete. In other words, the sequence of plans required to execute task τ may not be fully defined. This allows adjustments to be made if the state of the world changes due to outside influence (i.e. other agents in the world). The frequency of plan adjustments can be tuned for optimal performance. This tuning reflects a tradeoff between reliability and timeliness. If a task is defined in very precise terms (i.e. the sequence of plans that constitue the task is described explicitly), its sequence of plans must be accurate and robust to changes in the environment. In this case, the sequence of plans must be updated frequently resulting in a possible degradation of the planning sub-system's performance. Alternatively, if timeliness is more important than precision, the plan sequence will be updated less frequently but will be less robust to changes in the environment. The loss in robustness means that the overall mission is more likely to fail. Due to the adversary nature of the robotic soccer domain, a new plan to accomplish the specified task will be generated at each simulation step; therefore the system will err on the side of reliability.

4.2 The Supervisory Coach Component

The SimCanuck team can be supervised by a human coach during game play via a graphical user interface that allows the supervisor to specify missions and communicate these to the robots on the field. The nature of the interaction between the coach and the robotic team is one of supervisory control. If left alone, the team of robots will be able to accomplish the robotic soccer task (i.e. play the game). However, they should also be able to benefit from a coach's advice and demonstrate an improved performance. The supervisor communicates with the agents via a channel dedicated for on-line coach information built into the official RoboCup Soccer Simulation Server.

Originally, the system was intended to allow the coach to interact with the team on many different levels using both traded and shared control. This is not currently possible however due to limitations in the coach language used for communication. To enable the operator to exert direct control over agents on the robot would require a communication channel with nearly no latency. This limits the method of interaction between the coach and the team to shared or supervisory control.

Another goal of the coaching system is to allow the supervisor to give advice to sub-groups of the team as well as to the team as a whole. The size of a group could range anywhere from a one to eleven robots. The characteristics of the group or sub-group under the supervision of the coach will vary depending on the addressing scheme used when providing advice. There are two main categories for advice addressing: implicit and explicit.

Implicit addressing is achieved by the client-side analysis of the contents of the advice provided by the coach. For example offensive-play advice will only be interpreted by the sub-set of agents that are actively playing offense. In this case the allocation of tasks is achieved through the cooperative behaviours of the multi-agent society.

Explicit addressing is achieved by manually specifying one or more recipients for the given advice. The degree of control the operator has over a group can vary depending on the size of the recipient list. If only a single agent is specified as the recipient, the operator can exert a more direct control on the robot's behaviour by issuing an action command. However more abstract advice may be required if the recipient list is made up of a set of robots; in most cases, it would be useless to have all the robots in the group running the exact same pattern of movements unless they are performing formation tasks. The coach should be able to adjust the type of advice being provided depending on the size of the group selected as the recipients of the control information and what task should be accomplished (i.e. more specific tasks may be better suited to smaller groups). This allows a small amount of variation in the level of control; although not to the extent desired as stated above.

4.2.1 Filtering Control Information

One of the challenges in teleautonomous multi-robots is that any given instruction from the operator may only be of interest to a small portion of the group. To give an example in the robotic soccer context, the goalie robot will not be interested in commands relating to scoring goals as these are not relevant to its role. Similarly, when its team is in possession of the ball, a robot will not be interested in marking an opponent but rather escaping a mark put on them by an opponent. The filtering of command information in the SimCanuck system is achieved using one of two mechanisms. The first mechanism has the operator explicitly label a mission specification as either offensive or defensive using a conditional statement. For example, to specify a mission that is valid only when the team is on offense,

the condition should be set to hold only when the team is in possession of the ball. The second mechanism for filtering control information uses the behaviour network to preclude actions that are not relevant to the current goal of the robots. This was explained in section 4.1.2 and illustrated by the execution of a dribble mission. Any given command will only be valid to a subset of the behaviours that make up each robot's controller; these relevant behaviours will likely be subsumed in a situation where their specific behaviour does not match the current offensive-defensive state.

4.2.2 Coach Graphical User Interface

Interaction with the coach interface is performed mostly by pointing and clicking on the field view. The coach interface works at a task-level control mode [83] which can be further sub-divided into two operating modes: *coach* mode and *play-book* mode.

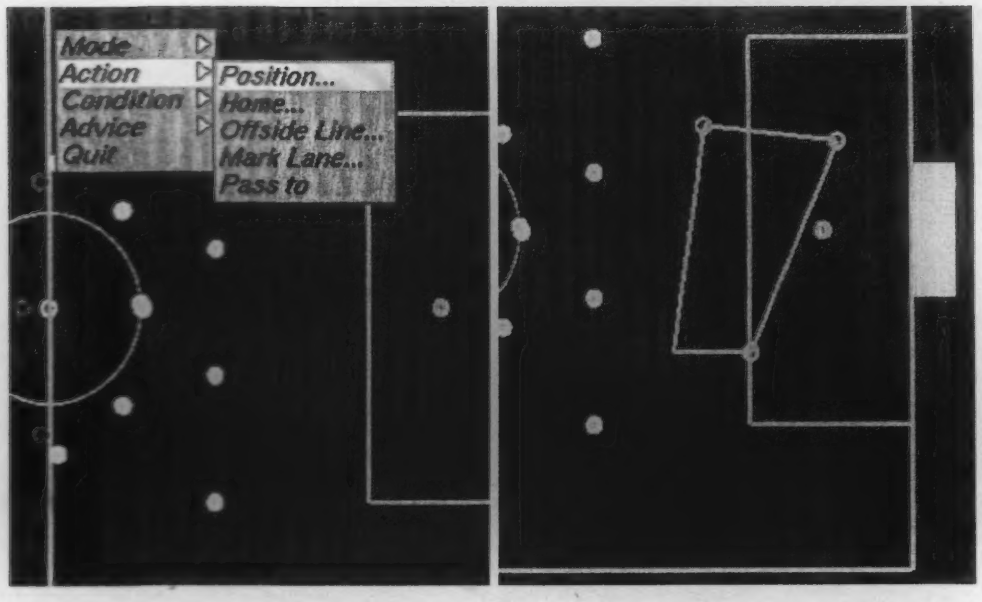
4.2.2.1 The *Play-Book* Operating Mode

The *play-book* mode provides the operator with a means of defining tasks and missions. In this operating mode, the user interface becomes a graphical configuration editor that allows the operator to construct mission specifications similar to the "cross-play" specification in Figure 4.5. Missions are constructed by selecting one or more objects of similar type (i.e. teammates, opponents or the ball) and using the context sensitive menus to select the desired advice. This is illustrated in Figure 4.6(a) where an action for a set of teammates is being selected. Some of the advised action types and condition types require a region to be specified. Once an advisory element of this type is selected, the interface is immediately put into the *Region Editor* mode to allow the appropriate region to be selected. Once the region is selected, the interaction mode is reset to the previously active mode. Region selection is illustrated in Figure 4.6(b).

The operator can switch to *play-book* at any time during the course of the game. This allows the operator to create new mission specifications that are specialized to the current situation and taylored for the behaviour of the current opponent. The set of mission specifications is saved and indexed using a mnemonic string, e.g. *cross-play* as a key. The mission set can also be stored to disk and re-loaded for use during a later game.

4.2.2.2 The *Coach* Operating Mode

The *coach* mode provides the operator with a means of assigning tasks and missions either defined in this session, in *play-book* mode, or in a previous session and loaded from a file. This feature can be useful as it allows operators to build up a library of pre-set missions and tasks that can be re-used or shared with other operators. Being able to store definitions is important as some time may be required to fully define a mission or task which would complicate the operator's job if a task or mission which is used multiple times must be



(a) Advising a new action.

(b) Region editor

Figure 4.6: Screen shots of the coach's graphical user-interface.

defined each time it is used. This also allows the operator to define a small library of tasks and missions before the start of a game; this helps reduce the operator's cognitive load while the game is being played.

Missions are communicated to the robots and are addressed either explicitly or implicitly. Explicit addressing is the responsibility of the operator and consists in assigning one or more identifiers to the control message so that it will be sent only to the specified robots. Operator commands are explicitly addressed to specific robots on the field by clicking on the desired recipients.

Implicit addressing, or task allocation, is the responsibility of the robots. When a mission specification is sent to the team without being explicitly addressed (i.e. the list of recipients is not selected by the operator), the agents on the field determine how best to allocate the tasks among the group. This is achieved by the behaviour controller which selects the task which best fits the agent's current situation and satisfies the constraints of the joint-task or mission.

Task allocation may also be required if a command is explicitly addressed. For example, a task may be distributed among a subset of the group selected by the operator. This may occur if a behaviour of higher priority than that issued by the coach is active in one of the agents in the selected group. For example the coach may designate a group of three robots

to position themselves around the opponent's penalty area when the ball is deep in that end of the field. However, if one of the designated agents is the closest teammate to the ball, the behaviour that guides the robot to chase the ball will likely subsume the positioning behaviour causing the agent to ignore the advice of the coach. The balance of the group of designated robots will most likely continue to follow the advice of the coach. Therefore, if the set of robot identifiers is greater than one, then task allocation may be performed at the agent level within this sub-group of the team.

Mission specifications may also be constructed while in *Coach* mode; however it is preferable to construct task specifications a priori and modify them as needed due to the time-critical nature of the robotic soccer environment. Because the soccer simulation is carried out in real time, it can be difficult to be able to analyse the game and define an appropriate task and issue it while it is still applicable and useful. This problem is further exacerbated by the restriction that coach messages are delayed by a number of cycles or discrete time steps; fifty by default. Each agent on the field will have executed fifty actions before the coach advice is received, therefore it is advantageous to take more time to carefully define tasks and missions so that their useful lifetime can be extended.

Chapter 5

Actions and Behaviours

In mathematics, symbolic logic serves as "a precise criterion for determining whether or not a given instance of mathematical reasoning is correct"[75]. Many of the principles in symbolic logic also have general application in domains other than mathematics. However, its use in many fields of human endeavor such as politics and salesmanship can be seen as having little value. This fact has prompted the creation of new logics, or variations on symbolic logic, to provide useful formalisms in other domains. The utility of such criterion for determining correctness of multi-agent systems is significant.

Models in science are stylised, abstract representations of a part of reality. Scientific discovery primarily creates models that represent a phemonemon being studied. Such models can be formal mathematical systems or practical and simple representations of a problem. For example, Newtonian physics is a mathematical model which allows the precise positions of objects to be determined based on the forces that act on them. Alternatively, models in biology consist of abstractions to communicate information about complex biological processes such as the reproduction and evolution of an organism. In each case, models are used to achieve different ends.

A real world phenomenon will be translated into a more abstract model which can be manipulated (by algebraic transformations, numerical computations or simple logical inferences) to obtain results that help better understand it and possibly predict its future behaviour. Operations of this nature allow astronomers to accurately predict the positions of the planets in our solar system.The utility of a model is to be more explicit and simpler to understand and manipulate than the real world phenomenon which it represents. A great number of details which are irrelevent to the system being studied may be eliminated, allowing researchers to concentrate on a more specific problem.

This section will describe a formalism that can be used to model reactive multi-agent systems. This formalism will, in subsequent chapters, be used to model the behaviour of the SimCanuck robots at the individual level as well as the collective level. This formalism is based on symbolic logic and uses elements and techniques from discrete mathematics.

Before the symbolic formalism is presented, however, some of the basic elements of discrete mathematics which are relevent to it will be defined.

5.1 Basic Definitions

Set Sets can be defined simply as collections of objects. Each set can be identified by a symbolic name which consists of a string of alpha-numeric characters with optional subscripts or superscripts. The objects contained in a set are called its elements and can be of any nature as long as they obey the following criteria:

1. Each object must be clearly defined
2. Each object in the set is distinct from every other object in the same set.

The empty set is the collection which contains no objects and is denoted by the symbol ϵ. The notation to represent a set consists of a comma separated list of symbols representing objects enclosed in curly braces. For example, the following is a set of integers:

$$\{1, 2, 3, -5\}$$

Note that the order of the elements of a set have no effect on the definition of the set. An object will either be a member of a set or not, no information about order is given. Two sets will be equal if they represent the same collection of objects; therefore the sets $\{1, 2, 3\}$ and $\{3, 2, 1\}$ represent the same set of integers in the range $[1, 3]$. Sets are clearly defined objects and as such can themselves be members of a set.

Member Given a collection of objects forming a set, any object that is part of the set is called a *member* of the set. An object x can be denoted as a *member* of a set named A using the expression $x \in A$.

Singleton Set A *singleton* set is a collection of exactly one object. The sets $\{1\}$ and $\{\epsilon\}$ are singleton sets or singletons as they have only one element. The first singleton is a collection containing a single integer and the second one is a collection containing the well defined object which is the empty set.

Subset and Superset Subsets and supersets can only be defined relative to another set. A set A will be a *subset* of another set B if and only if all of the elements of A are also members of B. Similarly, B will be a *superset* of A if and only if A is a *subset* of B. The fact that A is a *subset* of B and B is a *superset* of A can be denoted using the expression $A \subseteq B$.

Proper Subset and Superset A subset, A, of another set B will be a *proper subset* if and only if there exists an element in B which is not a member of A. Similarly, B will

be a *proper superset* of A if and only if A is a *proper subset* of B. The fact that a set A is a *proper subset* of B and B is a *proper superset* of A can be denoted using the expression $A \subset B$.

Power set The *power set* of a set A consists of a collection of all possible subsets of A. For example, given $A = \{1, 2, 3\}$, the power set of A is denoted as follows:

$$P(A) = \{\{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}\}$$

Ordered_pair An ordered pair of objects consists of a first object and a second object, which may or may not be distinct. Two ordered pairs are equal if their first and second elements are the same objects. Ordered pairs are different from sets of two elements because the order of the objects in the ordered pair is important and an ordered pair may contain two of the same object. The notation for ordered pairs is a comma separated list of two objects enclosed in angled brackets. The following are valid ordered pairs: $< a, b >$,$< a, a >$ and $< b, a >$. Note that the first and second example represent distinct ordered pairs (the order of the objects is different).

Finite sequence A *finite sequence* or *sequence* of objects consists of an ordered sequence of n objects where n is a non-negative integer. An ordered pair is also a finite sequence where $n = 2$. A sequence of length $n = 0$ (i.e. containing no elements) is defined as the empty sequence. Sequences are clearly defined objects and can therefore themselves be members of a set or sequence. This fact allows sequences of length $n > 2$ to be represented as nested ordered pair. For example, the sequence $< a, b, c, d >$ can be defined as the following ordered pair:

$$< a, < b, < c, < d, <>>>>>$$

This is how the Lisp programming language expresses lists. The first element of each pair is an element of the sequence and its nesting depth is its position in the sequence. The sequence $< a, b, c, d >$ is therefore expressed as a pair whose first element is the object a and the second object is the pair representing the remainder of the sequence. This second pair has the object b as its first element and the pair representing the remainder of the sequence as its second object and so on.

Function A function can be defined as a mapping from objects in a set A to objects in a set B. In other words, a function is a class, or set, of ordered pairs F with the property that the first object in each ordered pair will be a member of the set A, which is the function's domain, and the second object in each ordered pair will be a member of the set B, which is the codomain. The mapping from objects in set A to objects in set B must be well defined and deterministic. This imposes the following

restriction on F that no two ordered pairs will have the same object as their first elements. Thus for every $a \in A$, there is exactly one pair $< a, b > \in F$. A function may therefore be represented as an ordered triple (i.e. a finite sequence with $n = 3$ elements) $< A, B, F >$where A is the function domain, B is the function codomain and F is the set of ordered pairs which defined the mapping. In most cases, every member of the domain set A will be a member of an ordered pair in F. If this is the case then F represents a total function. The converse of this is that there exists an element $a \in A$ such that there does not exist a pair $< a, b >$in the set F.

The terms *mapping*, *map*, *operation*, *operator*, *transform*, *transformation* may be used interchangeably with the term *function*. The definition of a function will often be denoted by a name made up of a string of alpha-numeric characters with optional subscripts and superscripts. To define a name as a function and indicate its domain and codomain, the notation $f : A \longrightarrow B$ is used and is read as "f is a function from the domain A to the codomain B".

5.2 Modeling the Subsumption Network

Rodney Brooks introduced the subsumption architecture [15] as a new structure for controlling mobile robots. This architecture consists in designing the controller using layers that operate at increasing levels of competence. The controller is therefore composed of many layers which represent the core competencies of the robot. Each layer will represent an instance of a fairly simple computational machine. The operation of each controller is simple; for some input stimuli, an output will be obtained, which usually consists of commands to send to a robot's effectors. Each component can therefore be represented by a black box with any number of input lines and any number of output lines. The deeper reasoning of the controller emerges from the intreaction of the different layers of the controller as opposed to having the components themselves perform this complex task.

Using the elements of discrete mathematics, defined in section 5.1, a single behaviour module in a subsumption network can be defined using a function:

$$f_{\text{module}} : I \longrightarrow O$$

or

$$f_{\text{module}} =< I, O, B >$$

where I is a set of input stimuli, O is the set of operations that the module is permitted to execute, and B is a mapping, from inputs to operations, that models the overt behaviour of the module. Each element of the set of inputs I can be expressed as a finite sequence of stimuli.

Each module in the subsumption network will only be active in specific situations. For example, a behaviour that guides a mobile robot to back away from obstacles will only be active in the presence of an obstacle. Therefore the functions that model a behaviour module will not necessarily be represented by a total function. In other words, the set of input stimuli which, for a given behaviour module, will map to an output operation will be a subset of all the stimuli available to the controller. Therefore each module can be expressed as a total function by modifying the definitions provided previously:

$$f_{\text{module}} : I_{\text{module}} \longrightarrow O$$

and

$$f_{\text{module}} =< I_{\text{module}}, O, B_{\text{module}} >$$

where B_{module} is the mapping from input stimuli to operations which is specific to this module, and I_{module} is the set of input stimuli whose elements participate in a mapping in B_{module}. Let the set of all input stimuli available to the controller be I. Then the input that is relevant to the module described previously is a subset of I, that is $I_{\text{module}} \subseteq I$. This has the advantage that each behaviour module will be interpreted as a total function. The disadvantage is, of course, that the set of input must be fully defined for the mapping to be complete.

Assume the existence of a machine that takes as input a stream of alphanumeric strings and generates operations to be performed on these strings and uses a layered control system based on the subsumption architecture. The set of strings that may be seen by the machine is named I_{alnum} whose elements are pairs of string elements. One of the functions of this machine is to concatenate two strings into a single one. This behaviour will consist of a single module in the subsumption network and be interpreted by the following function:

$$f_{\text{concat}} : I_{\text{concat}} \longrightarrow O$$

where $I_{\text{concat}} \subseteq I_{\text{alnum}}$ and the set of outputs, O, consists of a set of relations that represent operations. For the time being, the set O will consist of the following singleton $\{\text{concat}(x, y)\}$, whose sole element is the well defined operation of concatenating two strings. The function f_{concat} can therefore be defined using the following criterion:

$$\forall < x, y > \in I_{\text{concat}}, f_{\text{concat}}(< x, y >) = \text{concat}(x, y)$$

The set of mappings, B_{concat}, from I_{concat} to O can be represented as follows:

$$B_{\text{concat}} = \{<< x, y >, \text{concat}(x, y) > | \forall < x, y > \in I_{\text{alnum}}, < x, y > \in I_{\text{concat}}\}$$

5.2.1 The Notion of Subsumption

The most important element in the subsumption architecture is the notion of subsumption. This is the mechanism that allows behaviours of higher priority (i.e. of higher layers in the

hierarchy) to subsume behaviours of lower priority. One of the major difficulties in modeling the subsumption architecture is in representing this subsumption relationship using the symbolic logic formalism described thus far. For this purpose, we introduce the operator $\succ$ to denote the position of modules in the hierarchy. Assuming the existence of two behaviour modules f and g defined as follows:

$$f : I_f \longrightarrow O$$

and

$$g : I_g \longrightarrow O$$

the fact that the behaviour f will subsume g is denoted as $f \succ g$. The function that results from the application of the $\succ$ operator to the functions f and g can be expressed as follows:

$$(f \succ g) : (I_f \cup I_g) \longrightarrow O$$

or

$$< I_f \cup I_g, O, B_{(f \succ g)} >$$

where the set of mappings $B_{(f \succ g)}$ is defined as follows:

$$B_{(f \succ g)} = \{< x, y > | (\forall x \in I_f, y = f(x)) \vee (\forall x \notin I_f, y = g(x))\}$$

The function $(f \succ g)$ will effectively simulate the subsumption of one behaviour by another because of the fact that each module is represented by a total function. This mapping would be impossible if the modules were defined using partial functions following the intuitive behaviour of the system.

When a given behaviour subsumes another, in this case f subsumes g, it is not desireable for them to handle the same set of input stimuli; this would result in the behaviour $(f \succ g)$ being the same as f which defeats the purpose of subsumption. In the definition of the function $(f \succ g)$, each function f and g have their own set of inputs. If these were to be equal, $I_f = I_g$ then the behaviour of the function $(f \succ g)$ would be that of f because the function $g(x)$ would be undefined for any element which is not a member of I_f. Therefore the set of mappings for $(f \succ g)$ would be defined as follows:

$$B_{(f \succ g)} = \{< x, y > | (\forall x \in I_f, y = f(x)) \vee (\forall x \notin I_f, g(x) = \text{undefined})\} = B_f$$

This situation illustrates the case where a "short circuit" exists in the subsumption network causing that certain behaviour will never become active. For this reason, the condition that the input set of a behaviour which subsumes another should not be the input set of the module being subsumed.

As an example of the subsumption relationship, a new module will be added to the controller defined in the previous section that performs operations on a stream of alphanumeric strings. Suppose that another desired behaviour of the controller is to perform the

sum of the values of integers represented by strings. In other words, if the pair of strings encoutered both represent integers, their values should be summed. This behaviour will be encapsulated by the f_{add} module:

$$f_{\text{add}} : I_{\text{add}} \longrightarrow O$$

where the set I_{add} is a subset of I_{alnum} which is defined as the set of pairs of strings that are integers; clearly in this case $I_{\text{add}} \subset I$. Also, by the definition of the set I_{add} it can be established that $I_{\text{add}} \subset I_{\text{concat}}$ as the set of all pairs of strings is clearly a proper superset of the pairs of strings that represent integers. In order to add this behaviour to the controller, the set of valid outputs must be redefined to contain the new operation $\text{sum}(x, y)$:

$$O = \{\text{concat}(x, y), \text{sum}(x, y)\}$$

The function f_{add} will therefore be defined using the following criterion:

$$f_{\text{add}}(< x, y >) = \{\text{sum}(x, y) | < x, y > \in I_{\text{add}}\}$$

The set of mappings, B_{add}, from the set I_{add} to O can be expressed as follows:

$$B_{\text{add}} = \{<< x, y >, \text{sum}(x, y) > | \forall < x, y > \in I, < x, y > \in I_{\text{add}}\}$$

Due to the fact that $I_{\text{add}} \subset I_{\text{concat}}$, then the best candidate function to be subsumed is f_{concat}, otherwise the function f_{add} may never become active. This subsumption relationship is expressed as $f_{\text{add}} \succ f_{\text{concat}}$. The application of the subsumption operator on these two functions will result in the new function $(f_{\text{add}} \succ f_{\text{concat}})$ which is defined as:

$$(f_{\text{add}} \succ f_{\text{concat}}) : I_{\text{concat}} \longrightarrow O$$

In this case, the input set is I_{concat} because of the fact that $I_{\text{add}} \subset I_{\text{concat}}$. The function $(f_{\text{add}} \succ f_{\text{con}})$ is defined as follows:

$$(f_{\text{add}} \succ f_{\text{concat}})(< x, y >) = \begin{cases} \text{sum}(x, y) & \text{for } < x, y > \in I_{\text{add}} \\ \text{concat}(x, y) & \text{for } < x, y > \notin I_{\text{add}} \end{cases}$$

This function can be expressed more clearly using the set of mappings for f_{add} and f_{concat}. Given that $I_{\text{add}} \subset I_{\text{concat}}$, it is possible to conclude that the union of the sets B_{add} and B_{concat} would result in a function that is not injective (i.e. that each member of the codomain maps to a different member of the domain). Let the function f_{meconcat} represent the same function as f_{concat} with the exception that I_{meconcat} will be mutually exclusive to I_{add}; that is $I_{\text{meconcat}} \bigcap I_{\text{add}} = \epsilon$. The mappings for the function f_{meconcat} can therefore be defined as:

$$B_{\text{meconcat}} = \{<< x, y >, \text{concat}(x, y) | (< x, y > \in I_{\text{concat}}) \wedge (< x, y > \notin I_{\text{add}})\}$$

Using this definition, the set of mappings for the function $(f_{\text{add}} \succ f_{\text{concat}})$ can be expressed as:

$$B_{(f_{\text{add}} \succ f_{\text{concat}})} = B_{\text{meconcat}} \cup B_{\text{add}}$$

5.2.2 Using Outputs as Inputs

Another key aspect of the subsumption architecture is that some modules may produce input values for other modules. This phenomenon can be represented by the composition of functions which is defined as follows. Given a function g, which maps from the set A to the set B, and a function f, which maps from the set B to the set C, the composition of f and g, denoted by $f \circ g$, is defined as $f(g(x))$. In other words, the composition of f and g maps an element a in A to the element in C assigned to $g(a)$. Therefore if f and g represent modules in the subsumption network, then the output of the module g for some input will be taken as the input for the function f.

As an example, assume that the input for the controller described in the previous section emmanates from another module called *split*. The behaviour of this module is to take a stream of input strings and split each string at the first occurence of a *dash* character. For this example, the input set for the controller, I, will be redefined as the set of strings such that each string s, having the format "ALNUM+'-'ALNUM+", is an element of I. The expression "ALNUM+'-'ALNUM+" is a regular expression that represents all strings of one or more alphanumeric characters, followed by a dash and one or more alphanumeric characters. The module f_{split}will therefore be defined as follows:

$$f_{\text{split}} : I_{\text{split}} \longrightarrow O_{\text{split}}$$

where $I_{\text{split}} \equiv I$ and O_{split} is a set of pairs of alphanumeric strings. Notice that the output space of this module is different than that of the previously defined modules. This is because the output space of f_{split} must be the same as the input space for f_{add} and f_{concat}. The function f_{split} will be defined ass follows:

$$f_{\text{split}}(z) =< x, y >$$

where x is the substring preceding the dash in z and y is the substring following it; the separating dash is discarded. The controller resulting from the use of the *split* module to provide the input to the *concat* and *add* modules is represented as:

$$(f_{\text{add}} \succ f_{\text{concat}}) \circ f_{\text{split}}$$

or

$$(f_{\text{add}} \succ f_{\text{concat}})(f_{\text{split}}(i))$$

The parentheses are used to disambiguate the operator precedence. However, in general, we shall let the subsumption operator be of higher precedence than the composition operator. Therefore the expressions $f_{\text{add}} \succ f_{\text{concat}} \circ f_{\text{split}}$ and $(f_{\text{add}} \succ f_{\text{concat}}) \circ f_{\text{split}}$ are equivalent. On the other hand, the expressions $f_{\text{add}} \succ (f_{\text{concat}} \circ f_{\text{split}})$ and $f_{\text{add}} \succ f_{\text{concat}} \circ f_{\text{split}}$ are not.

5.2.3 Using Multiple Inputs and Outputs

The use of multiple outputs for each module should, in the opinion of the author, be discouraged as this makes the controller much more complex and difficult to understand. However, it is nonetheless possible to represent such a controller using the current formalism. Behaviour modules with multiple inputs may also be used and can be represented by this formalism.

First, modules having multiple inputs will be discussed as this is the simpler case. A module having more than one input can easily be represented by a function that has more than one argument. Such a function is represented as

$$f : A \times B \longrightarrow C$$

where the input set, denoted by $A \times B$, is a set of pairs which represents every possible combination of elements in A with elements from B. For example, assume that the behaviour *concat* defined previously performed the concatenation of two strings which originated from two different sources. The function f_{concat} can be modified to represent this operation as follows:

$$f^2_{\text{concat}} : I_{\text{concat}} \times I_{\text{concat}} \longrightarrow O$$

where the sets I_{concat} and O are defined as before, however the input strings are split between two input lines. The function definition for the f^2_{concat} operation is:

$$f^2_{\text{concat}}(x, y) = \text{concat}(x, y)$$

The exponentiation notation applied to the function, as denoted in this example, provides information about the number of inputs used by the module. In this case the f_{concat} function was modified to use two input lines and therefore is re-named as f^2_{concat} as a result.

The representation of modules with multiple inputs poses a problem when using the composition operator. If the input values for a function f^2, which takes two arguments, are provided by functions g and h respectively, the formal representation of this configuration will be ambiguous. The composition could be denoted as $f^2 \circ g \circ h$; however this representation can also be interpreted as the case where h provides the input to g which provides the input to f^2, clearly not the desired behaviour. Cases where the composition of functions with multiple inputs is required can be denoted using parentheses rather than the composition operator. Therefore the function f^2 with inputs from g and h will be denoted as $f^2(g, h)$, which should eliminate any possible ambiguity.

Another possible approach to allowing the composition of a function with multiple functions is to represent the input functions as an ordered sequence. In this case, the composition of the function f^2 with g and h, as defined previously, would be denoted as $f^2 \circ < g, h >$.

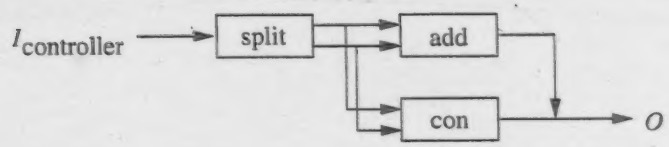


Figure 5.1: Diagram of a subsumption controller that performs operations on strings.

Formalising the use of multiple outputs is slightly more complicated than for multiple inputs. If a module has multiple outputs, each may provide the input for different modules. This has to be expressed clearly for the current formalism to be useful in modeling these cases. In symbolic logic, the domain of functions is in most cases a single set. This is the simplest and most expressive form for functions. Therefore, it is not desireable to have functions with outputs that span multiple sets (i.e. a function defined as $f : I \longrightarrow O \times O$). To avoid this situation, modules that provide multiple outputs should be expressed as multiple functions. For example, a function f with three outputs will be represented as three separate functions: f_1, f_2 and f_3. As an example, let the function f_{split}, defined previously, be redefined such that the input string will be split into two strings using the same approach; however the resulting two strings will be output on the first and second output lines appropriately. This module will be defined as follows:

$$f_{\text{split},1} : I_{\text{split}} \longrightarrow O_{\text{split}}$$

and

$$f_{\text{split},2} : I_{\text{split}} \longrightarrow O_{\text{split}}$$

The outputs of this module will provide the inputs to the *concat* and *add* modules. To accomodate this, the *add* module needs to be redefined as follows:

$$f^2_{\text{add}} : I_{\text{add}} \times I_{\text{add}} \longrightarrow O$$

where the sets I_{add} and O are defined as before with the input strings split between two input lines. The mapping from input to output for the function f_{add} will be defined as:

$$\forall x, y \in I_{\text{add}}, f^2_{\text{add}}(x, y) = \text{add}(x, y)$$

Using these definitions, the string operation subsumption controller can be denoted as $f^2_{\text{add}} \succ f^2_{\text{concat}} \circ < f_{\text{split},1}, f_{\text{split},2} >$ using the operator precedence rules described previously. This subsumption controller is illustrated in Figure 5.1.

5.3 Specifying and Reasoning about Multi-Agent Systems

Many agent theorists "present a framework for representing the mental state of agents and the physical states of the world and axiomatize the relationships between the various components (beliefs, goals, intentions, etc.) of the agent's mental state"[80]. Using these frameworks, success theorems which state that agents will achieve their intentions under certain conditions can be proved. However, as the complexity of the system increases, either through added complexity in a single agent's design or by the introduction of other cooperative or competitive agents into the system, it becomes unclear whether these theories can be used to reason about the actual behaviour of the system in a detailed way.

On the other hand, there exists formalisms for specifying and reasoning about concurrent systems, such as the Temporal Logic of Actions [53], that allow a proof of correctness of concurrent algorithms to be established. However, the use of these formalisms are at a disadvantage when specifying systems of multiple autonomous agents as they typically do not allow the explicit representation of actions in terms of their effects on the environment.

The formalism described previously in this chapter is intended to be used in conjunction with other frameworks to allow researchers to specify and reason about multi-agent systems.

5.3.1 Actions and their Effect on the Environment

The classic conception of an action is based on the notion of states and the transformation of states. For this representation to be of any use, the set Σ of possible states in an environment must be finite and expressible. An action will therefore be represented as a transition between states in Σ. This representation of actions is useful for proving the correctness of planning systems or agents that interact with their environments; however it suffers from several weaknesses which are most pronounced in the multi-agent case.

The classical conception of an action satisfies the following propositions[34]:

Staticity The environment is considered to be static in the sense that the only possible evolutions originate from an agent's actions.

Sequentiality The result of an action will be applied to the environment in a sequential fashion. If multiple actions occur simultaneously, their influence on the environment will be considered sequentially.

Universality Actions are described by their result on the environment. If multiple actions are required to bring the environment to its current state, it may not be possible to distinguish between the actions taken and the consequences of these actions.

These postulates weaken the representation of actions by limiting what can be modelled or by increasing the complexity of the model making the task of reasoning and analysis more difficult.

The proposition of staticity prevents environmental changes from being represented without the introduction of agent's into the system. For example, the flow of a river would have to be explicitly represented as an agent in order for it's influence on the environment to be modelled. The second proposition, sequentiality, makes it difficult to translate simultaneous actions into state transitions. If two agents perform complementary actions simultaneously, the net result may not change the current world state. However, because each action is represented as a state transition, the action of one agent must be applied before the action of another; possibly resulting in an incoherent state until all simultaneous state transitions have been applied. The final proposition, universality, increases the complexity of the model by requiring that every possible state, even those that result from an error or from an unexpected situation be represented; the influence of actions from multiple agents may temporarily result in unexpected situations until all actions are applied and the system achieves a coherent state.

To avoid the problems associated with the use of state transitions, actions should be represented as commands or operations. This is the most natural representation of actions when dealing with subsumption controllers. Using commands to represent actions circumvents the failings presented by the propositions of staticity, sequentiality and universality. Commands can be issued regardless of the current state of the environment; their effect will vary however, depending on the current state. Evolutions of the environment from influences outside the collective of agent's being modelled by the system are possible; the representation of actions as commands is not bound by the proposition of staticity. Commands do not need to be sequenced as state transitions do. If multiple actions are issued simultaneously by different agents, their effect on the environment may also be considered simultaneously. For example, commands may consist of forces applied to objects in the environment and simultaneous force vectors may be summed to determine the effective force being applied. Finally because commands are not coupled with their effects on the environment, distinguishing between actions and their conseqences is trivial.

Chapter 6

Analysis and Results

Chapters 3 and 4 described the overall design of the SimCanuck robot controller and provided motivations for certain decisions made in designing the operator interface and the behaviour hierarchy of the agents. The performance of the system as a whole however, has not yet been treated in this dissertation.

Many difficulties are presented when trying to empirically analyse the performance of MATS, particularly when dealing with highly random or adversarial environments where the outcomes of actions will rarely be repeatable. In these kinds of environments, the consequences of actions will be dependent on, possibly random, changes in the environment (e.g. variations in object trajectories due to noise) or influence of adversarial agents who's overt behaviours are not accurately predictable. These external influences, which make it difficult to attribute observable occurences to operator actions, correspond to independent variables which generally cannot be modeled or predicted with any degree of accuracy. The influence of these factors on the observable behaviour of the system can be minimised by 1) controlling the amount of randomness which occurs in the working environment and 2) controlling, as much as possible, the behaviours of the opposing agents. In the RoboCup Soccer Simulation Server it is possible to set the noise values associated with ball trajectory to effectively make these random variations insignificant. The behaviours of the opposing agents can also be controlled to some extent by re-using the same agent logics (assuming the agents used are not endowed with a learning algorithm of some sort). This variable cannot be fully controlled however as the reactive controller used by the SimCanuck robots is non-deterministic in nature and this non-determinism will likely "bleed" to the behaviour of opponent robots who are designed to react to the actions of their own opponents.

The skill of the human operator is also an important variable in determining the effectiveness of a MATS. Much like measuring intelligence, measuring an operator's level of skill is a highly subjective undertaking. Given identical scenarios, two operators may have differing approaches to solving the same problem and achieve the same measure of success. The differences in their approaches lies in their personal style in perceiving the current sit-

uation and reacting to unexpected events. An operator's skill level may also vary between successive uses of a supervisory control system. Factors such as fatigue and mood may cause the supervisor to advise the collective differently. Furthermore, the operator's familiarity with a MATS system will increase with experience allowing them to make more effective use of its supervisory facilities.

6.1 Empirical Analysis of the SimCanuck MATS

To test the SimCanuck MATS, a series of simulations were carried out between two SimCanuck robotic soccer agent squads; an unsupervised SimCanuck instance and a supervised "clone" of the same team. The motivation for this configuration is that the strongest of the two teams would tend to win most of the games and thus prove to be statistically better. For this comparison to be of any use, a statistical baseline must be established; this baseline is determined by playing a series of games between two unsupervised SimCanuck squads. In all, sixteen games were played and the outcomes of those games are shown in table 6.1. Each game consists of two halves of 3000 simulation cycles (roughly five minutes in real time). If neither team is victorious following these two halves of play, an additional two halves will be played using the golden goal rules (i.e first goal wins). Thus the probabilities of a win, loss or tie for a specific team following the end of regulation time will be $\frac{1}{3}$ for each case. In addition, once the game has reached extra time, the probability that a team will win in these extra frames will also be $\frac{1}{3}$. Let X be the random variable defined by the outcome of a soccer game in the RoboCup soccer simulation server, the probability density function of X is

$$f(X) = \begin{cases} \frac{4}{9}, & X = \mathrm{W} \\ \frac{4}{9}, & X = \mathrm{L} \\ \frac{1}{9}, & X = \mathrm{T} \end{cases}$$

Given the p.d.f of X it's easy to conclude that, on average, both teams will have the same number of wins over a period. However, during the trials from table 6.1 the team designated as the opponent had twice as many wins as the team designated as SimCanuck. This discrepancy can be attributed to the fact that most of the games were won by a single goal and that the outcome could easily have gone either way. Rather than just account for wins, losses and ties, the differences in score could assist in determining the more dominant team. Therefore the goal differential for each game was calculated and used as a measure of the relative strength of the two teams. In table 6.1 the final goal differential, following the sixteen trials, is -3 indicating that the opponent team scored three more goals than the team designated as SimCanuck. The average goal differential for these trials was -0.1875 and the standard deviation 3.5428. A more accurate interpretation of the data can be obtained by dropping the high and low value resulting in the second set of mean and variance values. The average goal differential of the normalised trial data is close to zero at -0.3571 with a

Game	Team 1 Score (unsupervised)	Team 2 Score (supervised)	Goal Differential
1	2	1	-1
2	2	0	-2
3	1	2	1
4	0	1	1
5	1	0	-1
6	2	1	-1
7	1	0	-1
8	1	1	0
9	1	2	1
10	2	1	-1
11	1	0	-1
12	2	0	-2
13	2	1	-1
14	5	2	-3
15	1	6	5
16	0	3	3
Totals	24	21	-3
Mean	1.5000	1.3125	-0.1875
Variance	1.3333	2.3625	3.5428
Mean*	1.2857	0.9285	-0.3571
Variance*	0.5274	0.8407	1.0897

Table 6.1: Results of the sample games played between two unsupervised instantiations of the SimCanuck robotic soccer system.

Game	Team 1 Score (unsupervised)	Team 2 Score (supervised)	Goal Differential
1	0	3	3
2	2	1	-1
3	0	0	0
4	0	0	0
5	3	0	-3
6	3	2	-1
7	4	8	4
8	2	4	2
9	1	4	3
10	0	5	5
11	6	2	-4
12	4	3	-1
13	2	5	3
14	0	1	1
15	2	5	3
16	6	9	3
Totals	35	52	17
Mean	2.3333	3.2500	1.0625
Variance	4.0952	7.4000	6.7292
Mean*	1.9230	3.0000	1.2143
Variance*	3.2435	6.6154	5.1044

Table 6.2: Results of the sample games played between a supervised instantiation of the SimCanuck robotic soccer system and an unsupervised one.

variance of roughly one goal. Therefore these numbers satisfy the intuitive expectation of the outcomes of opposing two identical teams.

Next a supervised SimCanuck squad was matched up against an unsupervised team. Once again sixteen games were played between a supervised and unsupervised team to compare the resulting statistics with those obtained previously using two unsupervised teams. The outcomes of the sixteen trials are shown in table 6.2. The final goal differential following the sixteen trials was 17 meaning that the supervised team scored seventeen more goals than the unsupervised squad winning nine of their sixteen games, seven by three goals or more, and losing only five. The average goal differential for these trials was 1.0625 with standard deviation of 6.7292. By dropping the games posting the highest and lowest goal differentials, normalised values for the mean and variance are calculated to be 1.2143 and 5.1044 respectively. The interpretation of these values is that the supervised team has outscored its opponent by more than one goal in most of its games with the actual total varying by, roughly, plus or minus five goals. This variance can be attributed to many factors including the supervisor's skill, the strategy employed and randomness. Certain strategies employed by a team of robotic soccer agents may be in-effective against other teams.

6.1.1 Qualitative Analysis of the Supvervised SimCanuck MAS

While carrying out the trials of the supervised SimCanuck MATS, several game statistics were taken down and observations were drawn about the behaviours of the robots with the influence of a supervisor's advice. For each trial, the same supervisor managed the team in order to limit variations as a result of the coach's personal abilities or style. Due to this fact, throughout the course of the experiments a particular strategy was found to work well and thus slight variations on this strategy were used for most of the games, effectively reducing the potential supervisor's potential workload. This will not necessarily be true in general however as this particular strategy may only be effective against the unsupervised SimCanuck team. Game strategies should constantly be modified to adapt to those of the opponents. Since the same opponent was used for each trial, drastic variations in strategy would be unncessesary. The fact that changes to the strategy were infrequent limits the possible analysis of the user interface that can be performed from these experiments. Some analysis about the useability of the interface can nontheless be carried out using the observations reported during the trials.

One important question that a supervisor must ask when coaching the SimCanuck robotic soccer team is how often to modify the advised strategy which is employed by the collective. Strategy changes will not necessarily have an immediate effect and it is important for the supervisor to decide how long to wait before concluding that a strategy is not effective and should be changed. If the supervised team is clearly being dominated, this decision is trivial. In other cases this decision may not be obvious; deciding if a particular strategy will increase the probability of goals being scored by the supervised team for example.

During experimentation, it was observed that changes to strategy occurred more often when the team being supervised was losing. This is a natural phenomenon since when the team is winning, the current approach to the game is by all intents and purposes correct, making changes unnecessary. As an example, in the eighth game of the trials, the supervised team was losing by a margin of two goals to none following the conclusion of the first half of play. Before the begining of the second half, the coach retooled the strategy to have the team play more aggressively (i.e. have more players in or near the offensive zone). This proved to be effective as the supervised team came back to win by a score of four goals to two in this case.

Despite the fact that changes to strategy should only be required in the event that a team is losing, these nonetheless occurred in most of the games played during the trials. For example, in the ninth game, which the supervised team won handily with a final score of four goals to one, the supervised team took an early lead in the first half. Thus the strategy remained unchanged at the start of the second half. However in the second half of play, the

unsupervised team had much more ball possession and was threatening to score. Since the supervised team had already amassed a comfortable lead, the strategy was modified to have the agents load up the defensive zone making it more difficult to move the ball into scoring position. The joint task, in this case, was to position the players in the defensive area and to attempt to keep the ball away from the goal area.

Because of limitations in the communication language constructs which prevent the specification of plans explicitly, one action at a time, low level commands could not be sent to the robots preventing direct control of individual robots. The ability to exert this kind of control could be advantageous in certain situations. For example if a robot faces an un-anticipated situation and does not deal with it appropriately, the supervisor could intervene and guide the robot to accomplish the required task. This level of control would increase the supervisor's cognitive load dramatically as much of the low level details that are better handled by the robot's AI would then have to be considered by the human. For these reasons, only abstract advice was used by the supervisor when coaching the SimCanuck team. It was therefore important to observe how the MATS reacted to the supervisor's advice.

As described previously, when high level missions and tasks are advised, the SimCanuck MATS is responsible for the allocation of these to members of the collective as necessary. In the sixth trial game, the supervisor provided positioning advice to spread the robots out on the field in order to make it more difficult for the opponents to cover all the players on the field. The agents would mostly play around the positions illustrated in Figure 6.1. This configuration allows the SimCanuck team to pass the ball around to an open player and make progress towards the opponent's goal in this way. The white arrows in the figure show how the ball was moved around from one player to the other to the goal. Note that no differentiation is made between ball movements as a result of a pass, a shot or a robot dribling the ball to along the specified path. This strategy proved to be only marignally effective as the supervised team ended up loosing this game by a margin of three goals to two.

The seventh game proved to be more productive for the supervised team as they managed to score eight goals to the opponent's four. The positioning and ball movement of the supervised team is illustrated in Figure 6.2. In this game, the forwards were positioned a little closer to the defensive end of the field than in the previous game yet they were close enough to the opponents goal to be effective on offense. This configuration made it easier for the defenders and mid-fielders to move the ball to the forwards because the distance between these two groups was shorter than in the previous game and therefore passes were less likely to be intercepted by the opponents.

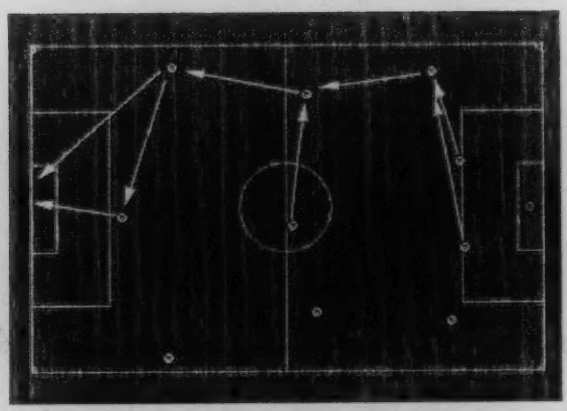


Figure 6.1: Game six ball movement overview.

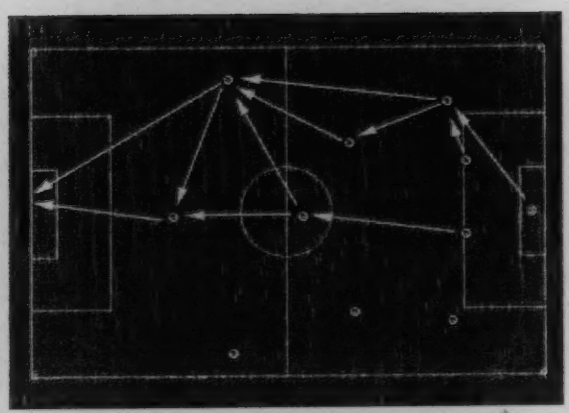


Figure 6.2: Game seven ball movement overview.

6.2 Theoretical Analysis of the SimCanuck MATS

The previous section provided an analysis of the empirical data gathered during a series of trials using the SimCanuck MATS. The results obtained helped to support the thesis that a supervised multi agent system will be more effective than a similar fully autonomous system. This section will present a theoretical analysis of the system in an attempt to explain how the system can achieve such an improvement.

Chapter 5 presented a formal model to assist in the validation of systems composed of multiple reactive agents. This model will be used to formally validate the MATS frameworks used by the SimCanuck soccer agents. The following discussion will be presented using a simplified tele-autonomous multi-robot system similar to the SimCanuck system. This simplified version of the SimCanuck MATS will not take into consideration the actions of the opponents and will deal with a strictly discrete state space. These are severe limitations as the actual robotic soccer domain 1) has non-discrete possibilities for actions and 2) will typically introduce an adversary which will try to prevent the successful completion of joint-goals. However this model should be sufficient to illustrate how the SimCanuck framework actually works.

6.2.1 Modeling the SimCanuck Behaviour

Let $\mathcal{L}$ be a first order language whose alphabet of symbols contains the constants a_i $(i > 0)$, the variables x_i $(i > 0)$, the collection of relations A_i^n $(i > 0, n > 0)$ and the collection of functions f_i^n $(i > 0, n > 0)$. The following will define an interpretation of $\mathcal{L}$ in a grid world environment Γ.

The grid-world environment consists of an $m \times n$ grid within which mobile robots move around and interact. The grid environment can be defined as a set of 2-tuples that describe its x-y coordinates. Let the symbol $\mathcal{G}_{m,n}$ represent the set of coordinates that define a grid that is m places wide and n places high. The set $\mathcal{G}_{m,n}$ can therefore be defined as:

$$\mathcal{G}_{m,n} = \bigcup_{j=1}^{n} \bigcup_{i=1}^{m} \{< i, j >\}$$

which roughly translates to the following matrix:

$$\begin{bmatrix} < 1,1 > & < 2,1 > & \cdots & < m,1 > \\ < 1,2 > & < 2,2 > & \cdots & < m,2 > \\ \vdots & \vdots & \ddots & \vdots \\ < 1,n > & < 2,n > & \cdots & < m,n > \end{bmatrix}$$

Two types of objects will occupy a grid $\mathcal{G}_{m,n}$; robot and ball objects. Robots will belong to a specific group, or collective, which is represented by a constant g that is a unique identifier of the collective to which the specified robot belongs. A robot object, belonging to a specified group g_i and identified by j, will be represented by the relation robot(g_i, j)

where j is the number of the robot. The robot's number is unique only to its group; in other words, given two robots robot(g_i, j) and robot(g_k, l), they will represent the same robot if, and only if, $j = l$ and $g_i = g_k$; otherwise they represent different robots. The set of all robots that occupy a grid $\mathcal{G}_{m,n}$ is represented by the symbol R.

Ball objects in a grid $\mathcal{G}_{m,n}$ exist only as individuals and therefore are not part of a group as is the case with robot objects. Ball objects are therefore represented as simple constants $b1, b2, \ldots$, which uniquely identify the ball object in the current instantiation of the grid world. The set of all ball objects that occupy a grid $\mathcal{G}_{m,n}$ is represented by the symbol B. The ability to model multiple ball objects is provided for completeness of the model.

The "at" relation is used to localise objects in the grid world. This relation has an arity of two and relates a specified object to an x-y coordinate in the grid world. Therefore the relation at(robot$(g_i, j), < x, y >)$ indicates that the robot identified by j which belongs to the group g_i is located at position $< x, y >$ in the grid.

The inclusion of mobile objects into the grid implies the concept of state which will now be defined. Let Σ represent a set of states $\{s_1, s_2, s_3, \ldots\}$ where a single state s_t is defined as:

$$s_t = \{\text{at}(b1, < x_1, y_1 >), \ldots, \text{at}(\text{robot}(g_1, 1), < x_2, y_2 >), \ldots, \text{at}(\text{robot}(g_2, 1), < x_3, y_3 >), \ldots\}$$

Each state s_t is itself a set whose elements are relations that describe the positions of objects in the environment. For example, in state s_t the relation at$(b1, < x, y >)$ represents the position of the ball object, identified by the constant $b1$, as $< x, y >$ for state s_t where x and y are arbitrary values in the set of positive integers $\mathbb{Z}^+$. Note that the coordinates are represented as ordered pairs in order to differentiate the coordinates $< x, y >$ and $< y, x >$ which is not possible using sets.

Given the set of ball objects B, robot objects R, grid points $\mathcal{G}_{m,n}$ and valid states Σ, the domain $\mathcal{D}_\Gamma$ of an interpretation of the language $\mathcal{L}$ in Γ can be defined as:

$$D_\Gamma = B \times R \times \mathcal{G}_{m,n} \times \Sigma$$

Based on this definition of the domain of Γ, the set of functions can now be defined for the language $\mathcal{L}$ in Γ. The first function f^2_{dist} has two places for arguments and defines the distance between two points in grid $\mathcal{G}_{m,n}$.

$$f^2_{\text{dist}} : \mathcal{G}_{m,n} \times \mathcal{G}_{m,n} \longrightarrow \mathbb{R}^+$$

where $\mathbb{R}^+$ is the set of all *real* positive numbers.

$$f^2_{\text{dist}}(< x_1, y_1 >, < x_2, y_2 >) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{6.1}$$

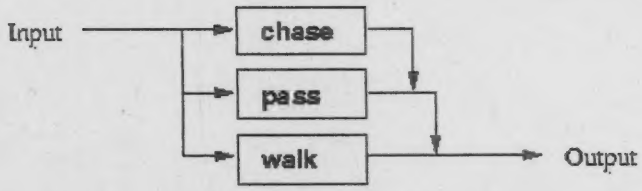


Figure 6.3: A simple behaviour network for the grid-world robots.

The second function that will be needed to express the changes to the grid-world is the "closest" operation which, when supplied with an object descriptor, allows the next closest object to it to be found. Let the relation $\leq$ be the interpretation of the predicate letter A^2_{leq} in $\mathcal{L}$. The *closest* function can now be defined using the distance function 6.1 and the predicate A^2_{leq}:

$$f^2_{\text{closest}} : B \times \Sigma \longrightarrow R$$

The function f^2_{closest} finds the robot that is closest to the ball object specified as the first argument in the state specified as the second argument. This function is calculated as follows:

$$f^2_{\text{closest}}(b_i, s) = x_2$$

where $(x_2 \in R)$, $(\text{at}(b_i, x_1), \text{at}(x_2, x_3) \in s)$, $(x_1, x_3 \in \mathcal{G}_{m,n})$ and

$$x_2 = \text{at}(g_j, k), (\forall \text{at}(x_4, x_5) \in s)(\neg((x_4 \in R) \rightarrow (\neg(A^2_{\text{leq}}(f^2_{\text{dist}}(x_1, x_3), f^2_{\text{dist}}(x_1, x_5)))))$$

The remainder of the functions in Γ are used to model the reactive controller that regulates the behaviour of each robot in the environment. Before formally defining these functions, a simple controller based on the subsumption architecture will be described. Each robot's controller is made up of a network of three behaviours, *chase*, *pass* and *walk*. The behaviour network composed of these three behaviours is illustrated in Figure 6.3. These behaviours are very specific to the grid world and the controller is very simple, however this simplifies the process of drawing observations about the subsumption architecture in general.

Each behaviour in the network has a very specific function. The *chase* behaviour, when activated, will direct the robot towards the ball's position; this behaviour will be active only when the ball is located in the robot's immediate neighborhood. The *pass* behaviour will, when the robot is in possession of the ball, pass it to the closest robot that belongs to the same collective. This behaviour will be active *only* when the robot is in possession of the ball (i.e. when its coordinate and that of the ball are the same) and if there is a target available for the pass. Note that in the case where there is only a single robot in the grid-world, the pass behaviour will never be active as there will be no available targets.

The Final behaviour, *walk*, directs the robot to perform a random walk of the grid-world. This is the default behaviour and is subsumed by both the *chase* and *pass* behaviours as illustrated in Figure 6.3.

Each of the behaviours described above can be represented by a function defined in a first-order logic. Using the notation introduced in Chapter 5, the formal definition of the controller can be expressed as $f_{chase} \succ f_{pass} \succ f_{walk}$. The input set for the controller is the domain D_Γ of the language $\mathcal{L}$, and the output is the set of commands, namely

$$O = \{\text{move(Obj,north)}, \text{move(Obj,south)}, \text{move(Obj,east)}, \text{move(Obj,west)}, \text{kick(Ball,Pos)}\}$$

This set can be further separated into movement and kicking commands

$$O_{\text{movement}} = \{\text{move(Obj,north)}, \text{move(Obj,south)}, ...\}$$

and

$$O_{\text{kicking}} = \{\text{kick(Ball,} < 1,1 >), \text{kick(Ball,} < 1,2 >), ...\}$$

The model used to represent actions is based on the notion of influence and reactions to influences. Therefore the commands in O will influence the future state of the environment. For example, discounting any other influence on the environment, the application of the command move(Obj,north) to the current state, s_t, of the environment will result in the state s_{t+1} where the object specified by *Obj* is located one grid area immediately above its position in state s_t. Movements to the *south*, *west* and *east* will results in displacements of one grid area below, to the left and to the right respectively.

The actual changes to the environment will be effected by the function *apply* which is defined as follows:

$$f^2_{\text{apply}} : O \times \Sigma \longrightarrow \Sigma$$

This function defines the influence of a single command to the state of the environment. It maps from the set of commands O and the set of world states Σ to obtain a new state in Σ. This weakens the representation of actions, as described in Section 5.3.1, as their influence on the environment must be considered sequentially therefore limiting the ability of the model to express multi-agent systems. In order to avoid this limitation, the *apply* function can be re-defined as follows:

$$f^2_{\text{apply}} : O^* \times \Sigma \longrightarrow \Sigma \tag{6.2}$$

where the set O^* is the power-set of the set of commands $P(O)$. Therefore the first argument of function 6.2 will be a set of commands whose influences must be applied to the current world state.

Definition of the Behaviour Modules

The first component of the controller, the *chase* behaviour, is defined as follows:

$$f^2_{\text{chase},\varphi} : \Sigma_{\text{chase},\varphi} \longrightarrow O$$

This function maps from the set of states $\Sigma_{\text{chase},\varphi}$ to obtain an appropriate command in O. The set of states that are used as input to the *chase* behaviour is a subset of the set of all states and is defined as follows:

$$\Sigma_{\text{chase},\varphi} = \{s | \forall bi \in B, \exists \varphi \in R, f^2_{\text{closest}}(bi, s) = \varphi\}$$

The input set is simply all the states where the robot represented by φ is the closest to a ball object. This set of input states will be unique for each robot as it is defined in relation to specific agents. The function mapping for the *chase* function is defined as

$$f^2_{\text{chase},\varphi}(s_t) = \text{move}(\varphi, \text{Dir})$$

The second argument in the *move* command, *Dir* is a placeholder for the direction that will move the agent closer to the ball's position. The actual value of the *Dir* will be either a horizontal or vertical direction guiding the robot to a position having a shortest distance to the ball. This movement is made assuming that the robot can only move one square at a time in a horizontal or vertical direction. If, for example, the ball is positioned diagonally from the robot such that the horizontal displacement and the vertical displacement result in the robot being the same distance from the ball, a direction is chosen at random. This is illustrated in Figure 6.4.

The next component in the controller is the *pass* behaviour. The formal definition of this behaviour is as follows:

$$f^1_{\text{pass},\varphi} : \Sigma_{\text{pass},\varphi} \longrightarrow O$$

This function maps from the set of states $\Sigma_{\text{chase},\varphi}$ to obtain an appropriate command in O. The set of states that are used as input to the *pass* behaviour is a subset of the set of all states and is defined as follows:

$$\Sigma_{\text{pass},\varphi} = \{s | \forall b \in B, \exists \rho \in R, (\rho = \varphi) \wedge (\text{at}(\rho, x_1) \in s) \wedge (\text{at}(b, x_2) \in s) \wedge (\text{at}(\varphi, x_2) \in s)\}$$

This defines the input set of the *pass* behaviour as all states where the robot, specified by φ, occupies the same position as a ball object b and there exists another robot in the grid other than φ. As is the case with the input set for the *chase* behaviour, this set of input states will be unique for each robot as it is defined in relation to a specific agent. The function mapping for the *pass* behaviour is defined as

$$f^1_{\text{pass},\varphi}(s_t) = \text{kick}(\text{Ball}, x_1)$$

where there exists a robot ρ such that $\text{at}(\rho, x_1) \in s_t$, $\rho \neq \varphi$ and ρ is the closest teammate to φ. The first argument in the ***kick*** command, ***Ball***, is a placeholder for the ball that is currently kickable by the robot φ; in other words Ball $= b$ where $\text{at}(b, x_2) \in s$ and $\text{at}(\varphi, x_2) \in s$.

The final module in the controller is the ***walk*** behaviour. This is the fallback behaviour and will therefore have the lowest priority of all the previous behaviours. The input set of this behaviour will be the input set of the entire controller less the input sets of the two modules of higher priority than ***walk***. The input set, Σ_{walk} can therefore be defined as:

$$\Sigma_{\text{walk},\varphi} = I \bigcap \overline{\Sigma_{\text{chase},\varphi} \bigcup \Sigma_{\text{pass},\varphi}}$$

The ***walk*** function will therefore be defined as

$$f^1_{\text{walk},\varphi} : \Sigma_{\text{walk},\varphi} \longrightarrow O$$

This module represents a random walk behaviour, therefore the coordinates of the robot, following this behaviour, will change randomly with given discrete probabilites. The result will be that either the x or y component of the robot's current position will be incremented or decremented by one. The possibilities for the direction of the random walk are dependent on whether the desired movement is valid according to the constraints of the environment. These constraints include the bounds of the grid or any physical obstacles, such as another robot, which prevent movement in a specific direction. To determine if a command is valid in the current state, the function f^2_{valid} must first be defined.

$$f^2_{\text{valid}} : O_{\text{movement}} \times \Sigma \longrightarrow \{0, 1\}$$

This function maps from the set of movement commands O_{movement} and the set of states Σ to a subset of $\mathbb{Z}$ which contains the integer values 1 and 0. The validity function will evaluate to 0 if the movement is invalid, and evaluate to 1 otherwise:

$$f^2_{\text{valid}}(\text{move}(\varphi, \text{Dir}), s) = \begin{cases} 0 & \text{if movement is valid} \\ 1 & \text{otherwise} \end{cases}$$

Using the validity function, the probability distribution function for the random walk behaviour is calculated as

$$P(f^1_{\text{walk},\varphi}(s) = a) = \begin{cases} \frac{f^2_{\text{valid}}(\text{move}(\varphi,\text{north}),s)}{u(f^2_{\text{valid}}(\text{move}(\varphi,\text{north}),s))} & a = \text{move}(\varphi, \text{north}) \\ \frac{f^2_{\text{valid}}(\text{move}(\varphi,\text{south}),s)}{u(f^2_{\text{valid}}(\text{move}(\varphi,\text{south}),s))} & a = \text{move}(\varphi, \text{south}) \\ \frac{f^2_{\text{valid}}(\text{move}(\varphi,\text{east}),s)}{u(f^2_{\text{valid}}(\text{move}(\varphi,\text{east}),s))} & a = \text{move}(\varphi, \text{east}) \\ \frac{f^2_{\text{valid}}(\text{move}(\varphi,\text{west}),s)}{u(f^2_{\text{valid}}(\text{move}(\varphi,\text{west}),s))} & a = \text{move}(\varphi, \text{west}) \end{cases}$$

where

$$u(x) = 1 + f^2_{\text{valid}}(\text{move}(\varphi, \text{north}), s) + f^2_{\text{valid}}(\text{move}(\varphi, \text{south}), s) + f^2_{\text{valid}}(\text{move}(\varphi, \text{east}), s) + f^2_{\text{valid}}(\text{move}(\varphi, \text{west}), s) - x$$

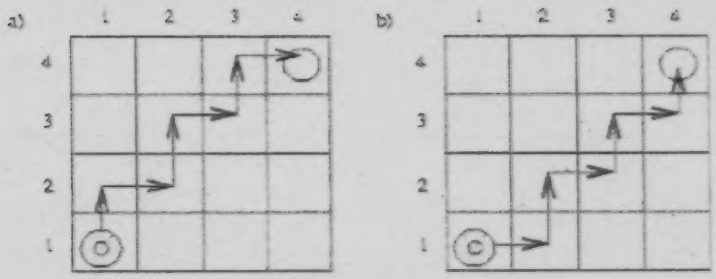


Figure 6.4: Examples of the execution of the *chase ball* behaviour in a grid $\mathcal{G}_{4,4}$ with a single robot and a single ball.

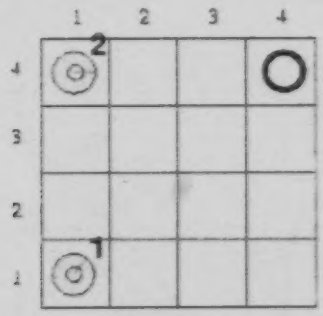


Figure 6.5: Initial state of the example grid world with a 4×4 grid, 1 ball object and 2 robots belonging to the same collective.

Having now formally defined the basic behaviours for the robots in the grid world, the subsumption controller can be represented by the function *behave*, defined as follows:

$$f^1_{\text{behave}} : \Sigma \longrightarrow O$$

The mapping for this function is defined as $f^1_{\text{chase}} \succ f^1_{\text{pass}} \succ f^1_{\text{walk}}$. In other words

$$f^1_{\text{behave}}(s_t) = (f^1_{\text{chase}} \succ f^1_{\text{pass}} \succ f^1_{\text{walk}})(s_t)$$

6.2.2 Toy Example Using a 4×4 Grid World with Tropistic Agents

The interpretation of the first order language $\mathcal{L}$ given in section 6.2.1 is general for any grid $\mathcal{G}_{m,n}$. The following notation will be used to represent a specific example of the above interpretation. Let $\Gamma^{i,j}_{m,n}$ represent the domain of a specific interpretation of $\mathcal{L}$ where m is the width of the grid, n is the height of the grid (as in the definition of $\mathcal{G}_{m,n}$), i is the number of ball objects that occupy the grid, and j is the number of robots. For now, assume that all robots are from the same collective or team. The following is a probabilistic model of the behaviours of the robots in a system $\Gamma^{1,2}_{4,4}$ with the start state illustrated in figure 6.5.

The set of commands for the current interpretation of the language $\mathcal{L}$ is the union of valid movement commands

$$O_{\text{movement}} = \{\text{move}(\text{robot}(r,1),\text{north}), \text{move}(\text{robot}(r,1),\text{south}), ...\text{move}(\text{robot}(r,2),\text{west})\}$$

and valid kicking commands

$$O_{\text{kicking}} = \{\text{kick}(b1, <1,1>), \text{kick}(b1, <1,2>), ..., \text{kick}(b1, <4,4>)$$

and can be expressed as $O = O_{\text{movement}} \bigcup O_{\text{kicking}}$. The power set of O is the set of all possible combinations of commands that can simultaneously influence the environment; the set of ***valid*** combinations of commands is a subset of this power set.

The formal definition of the initial state is given as follows:

$$s_1 = \{\text{at}(b1, <4,4>), \text{at}(\text{robot}(r,1), <1,1>), \text{at}(\text{robot}(r,2), <1,4>)\}$$

Given a sequence of desired actions, it is possible to calculate the probability of these actions actually taking place. For example, suppose the desired final state has robot$(r,2)$ in position $<4,4>$ and robot$(r,1)$ in some arbitrary position (i.e. the actions of robot 1 are not considered). To achieve this final configuration, the optimal sequence of actions would be for robot$(r,2)$ to walk 3 spaces east, or towards the ball. Because robot$(r,2)$ is closest to the ball object's position, its action will be defined as:

$$a_{1,\text{robot}(r,2)} = f^1_{\text{behave}}(s_1) = f^1_{\text{chase}}(s_1) = \text{move}(\text{robot}(r,2),\text{east})$$

While robot$(r,2)$ is moving toward the ball's position, robot$(r,1)$ will be performing a random walk in the grid. The action of robot$(r,1)$will be defined as:

$$a_{1,\text{robot}(r,1)} = f^1_{\text{beahve}}(s_1) = f^1_{\text{walk}}(s_1) = \text{move}(\text{robot}(r,1),\text{Dir})$$

where

$$P(\text{Dir} = x) = \begin{cases} \frac{1}{2} & x = \text{north} \\ \frac{1}{2} & x = \text{east} \end{cases}$$

The set of influences, $i_1 \in O^*$, that affects the state s_1 is defined as

$$i_1 = \{\text{move}(\text{robot}(r,2),\text{east}), \text{move}(\text{robot}(r,1),\text{Dir})\}$$

The next state of the grid environment, s_2, is obtained by applying this set of influences to the state s_1:

$$s_2 = f^2_{\text{apply}}(i_1, s_1) = \{\text{at}(b1, <4,4>), \text{at}(\text{robot}(r,1), x_1), \text{at}(\text{robot}(r,2), <2,4>)\}$$

where the variable x_1 is a placeholder used to represent robot$(r,1)$'s position. From state s_2 there is only a single direction in which robot$(r,2)$ can travel to reduce its distance to the

ball object in position $< 4,4 >$, therefore the next action that will taken by robot$(r,2)$ can be determined with certainty as move(robot$(r,2)$, east) and the probability that the system will reach state s_2 can be expressed as

$$P(s_1 \cap s_2) = P(s_1)P(s_2|s_1) = 1 \times 1 = 1$$

Therefore the system will achieve state s_2 with certainty.

The next two steps can be calculated similarly. The action taken by each robot from state s_2 is given as follows:

$$a_{2,\text{robot}(r,1)} = f^1_{\text{behave}}(s_2) = f^1_{\text{walk}}(s_2) = \text{move}(\text{robot}(r,1), \text{Dir})$$

and

$$a_{2,\text{robot}(r,2)} = f^1_{\text{behave}}(s_2) = f^1_{\text{chase}}(s_2) = \text{move}(\text{robot}(r,2), \text{east})$$

The resulting set of influences, $i_2 = \{a_{2,\text{robot}(r,1)}, a_{2,\text{robot}(r,1)}\}$, when applied to the state s_2 will result in a new state s_3:

$$s_3 = f^2_{\text{apply}}(i_2, s_2) = \{\text{at}(b1, < 4,4 >), \text{at}(\text{robot}(r,1), x_1\prime), \text{at}(\text{robot}(r,2), < 3,4 >)\}$$

with probability

$$P(s_2 \cap s_3) = P(s_2)P(s_3|s_2) = 1 \times 1 = 1$$

The actions taken from state s_3 are:

$$a_{3,\text{robot}(r,1)} = f^1_{\text{behave}}(s_3) = f^1_{\text{walk}}(s_3) = \text{move}(\text{robot}(r,1), \text{Dir})$$

and

$$a_{3,\text{robot}(r,2)} = f^1_{\text{behave}}(s_3) = f^1_{\text{chase}}(s_3) = \text{move}(\text{robot}(r,2), \text{east})$$

The resulting set of influences, $i_3 = \{a_{3,\text{robot}(r,1)}, a_{3,\text{robot}(r,2)}\}$, when applied to state s_3 will result in a new state s_4:

$$s_4 = f^2_{\text{apply}}(i_3, s_3) = \{\text{at}(b1, < 4,4 >), \text{at}(\text{robot}(r,1), x_1\prime\prime), \text{at}(\text{robot}(r,2), < 4,4 >)\}$$

with probability

$$P(s_3 \cap s_4) = P(s_3)P(s_4|s_3) = 1 \times 1 = 1$$

The final configuration s_4 with robot$(r,2)$ in possession of the ball and robot$(r,1)$ in some arbitrary position will therefore be achieved with a certainty (i.e. with probability $P_1 = 1$).

Suppose that in addition to having robot$(r,2)$ gain possession of the ball, which is positioned at $< 4,4 >$, the desired goal is to move the ball to position $< 4,1 >$. This can only be achieved by a pass from a robot in possession of the ball at $< 4,4 >$ to a robot who has positioned itself at $< 4,1 >$. Given that robot$(r,2)$ will chase the ball with probability 1, the probability of robot$(r,1)$ making its way to position $< 4,1 >$ will be the

limiting factor in the accomplishment of the goal. The probability of robot$(r,1)$ reaching position $<4,1>$ using the most direct path can be calculated as follows. The first action taken by robot$(r,1)$ was determined previously to be

$$a_{1,\text{robot}(r,1)} = \text{move}(\text{robot}(r,1), \text{Dir})$$

where the probability that Dir = east is $\frac{1}{2}$. Therefore the probability that robot$(r,1)$'s position in state s_2 will be $<2,1>$ is also $\frac{1}{2}$ and state s_2 will be expressed as

$$s_2 = \{\text{at}(b1, <4,4>), \text{at}(\text{robot}(r,1), <2,1>), \text{at}(\text{robot}(r,2), <2,4>)\}$$

Since robot$(r,2)$ will reach position $<2,4>$ in state s_2 with certainty, the probability of reaching this state will be dependent on robot$(r,1)$'s actions. Therefore the probabilty of reaching state s_2 is determined as

$$P(s_1 \cap s_2) = P(s_1)P(s_2|s_1) = 1 \times \frac{1}{2}$$

From state s_2, there are three valid movements available to robot$(r,1)$. Therefore a movement to the east by robot$(r,1)$ from state s_2 will be taken with probability $\frac{1}{3}$. This will result in a state s_3 where robot$(r,1)$ is located at position $<3,1>$ which is expressed as

$$s_3 = \{\text{at}(b1, <4,4>), \text{at}(\text{robot}(r,1), <3,1>), \text{at}(\text{robot}(r,2), <3,4>)\}$$

The probability of reaching state s_3 is

$$P(s_2 \cap s_3) = P(s_2)P(s_3|s_2) = \frac{1}{2} \times \frac{1}{3} = \frac{1}{6}$$

From state s_3, there are again three valid movements available to robot$(r,1)$. Therefore a movement to the east by robot$(r,1)$ from state s_3 will be taken with probability $\frac{1}{3}$ as before. This will result in a state s_4 expressed as

$$s_4 = \{\text{at}(b1, <4,4>), \text{at}(\text{robot}(r,1), <3,1>), \text{at}(\text{robot}(r,2), <4,4>)\}$$

where the probability of reaching state s_4 is

$$P(s_3 \cap s_4) = P(s_3)P(s_4|s_3) = \frac{1}{6} \times \frac{1}{3} = \frac{1}{18}$$

The final configuration s_4 with robot$(r,1)$ positioned at $<4,1>$ will be achieved with probability $P_2 = \frac{1}{18}$.

The final task to accomplish the multi-step mission of moving the ball to position $<4,1>$ is to pass the ball towards a player positioned at $<4,1>$. This must be done in two steps as the robot$(r,1)$ will continue to perform a random walk while the robot$(r,2)$ will be passing the ball towards it when it is in position $<4,1>$. However, this will be done with certainty, as the ball will be moved to position $<4,1>$ while robot$(r,1)$ will move either to position

$< 4,2 >$ or $< 3,1 >$ as these are the only two valid positions from $< 4,4 >$. The actions taken from state s_4 are determined as:

$$a_{4,\text{robot}(r,1)} = f^1_{\text{behave}}(s_4) = f^1_{\text{walk}}(s_4) = \text{move}(\text{robot}(r,1), \text{Dir})$$

where

$$P(\text{Dir} = x) = \begin{cases} \frac{1}{2} & x = \text{west} \\ \frac{1}{2} & x = \text{north} \end{cases}$$

and

$$a_{4,\text{robot}(r,2)} = f^1_{\text{behave}}(s_4) = f^1_{\text{pass}}(s_4) = \text{kick}(b1, < 4,1 >)$$

The application of the influence set $i_4 = \{a_{4,\text{robot}(r,1)}, a_{4,\text{robot}(r,2)}\}$ will result in two possible alternative states. These alternatives will be labeled as $s_5\prime$ and $s_5\prime\prime$ and are expressed as follows:

$$s_5\prime = \{\text{at}(b1, < 4,1 >), \text{at}(\text{robot}(r,1), < 4,2 >), \text{at}(\text{robot}(r,2), < 4,4 >)\}$$

and

$$s_5\prime\prime = \{\text{at}(b1, < 4,1 >), \text{at}(\text{robot}(r,1), < 3,1 >), \text{at}(\text{robot}(r,2), < 4,4 >)\}$$

In both of these states, robot$(r,1)$ becomes the closest robot to the ball's position. Therefore robot$(r,1)$ will chase the ball while robot$(r,2)$ will perform a random walk. The action taken by robot$(r,1)$ from state $s_5\prime$ is determined as:

$$a_{5,\text{robot}(r,1)} = f^1_{\text{behave}}(s_5\prime) = f^1_{\text{chase}}(s_5\prime) = \text{move}(\text{robot}(r,1), \text{south})$$

Alternatively, from state $s_5\prime\prime$, the action taken by robot$(r,1)$ is determined as:

$$a_{5,\text{robot}(r,1)} = f^1_{\text{behave}}(s_5\prime\prime) = f^1_{\text{chase}}(s_5\prime\prime) = \text{move}(\text{robot}(r,1), \text{east})$$

The action taken by robot$(r,2)$ will be the same from both states $s_5\prime$ and $s_5\prime\prime$ and is determined as:

$$a_{5,\text{robot}(r,2)} = f^1_{\text{behave}}(s_5\prime) = f^1_{\text{walk}}(s_5\prime) = \text{move}(\text{robot}(r,2), \text{Dir})$$

where

$$P(\text{Dir} = x) = \begin{cases} \frac{1}{2} & x = \text{west} \\ \frac{1}{2} & x = \text{south} \end{cases}$$

The application of the influence set $i_5 = \{a_{5,\text{robot}(r,1)}, a_{5,\text{robot}(r,2)}\}$ will result in to possible alternative states. These will be labeled as $s_6\prime$ and $s_6\prime\prime$ and are expressed as follows:

$$s_6\prime = \{\text{at}(b1, < 4,1 >), \text{at}(\text{robot}(r,1), < 4,1 >), \text{at}(\text{robot}(r,2), < 3,4 >)\}$$

and

$$s_6\prime\prime = \{\text{at}(b1, < 4,1 >), \text{at}(\text{robot}(r,1), < 4,1 >), \text{at}(\text{robot}(r,2), < 4,3 >)\}$$

	robot$(r,1)$				robot$(r,2)$			
n\m	1	2	3	4	1	2	3	4
1	0	$\frac{11}{36}$	0	$\frac{1}{18}$	0	0	0	0
2	$\frac{11}{36}$	0	$\frac{5}{36}$	0	0	0	0	$\frac{1}{6}$
3	0	$\frac{5}{36}$	0	0	0	0	$\frac{1}{6}$	0
4	$\frac{1}{18}$	0	0	0	0	$\frac{1}{6}$	0	$\frac{1}{6}$

Table 6.3: Listing of the probabilities of the final positioning of robots $(r,1)$ and $(r,2)$ after six steps using a simple subsumption based controller.

The team of robots r will therefore be able to achieve the joint-goal of taking possession of the ball at position $< 4,1 >$ with probability

$$\begin{aligned}
P(s_6) &= P(s_5\prime \cap s_6\prime) + P(s_5\prime\prime \cap s_6\prime) + P(s_5\prime \cap s_6\prime\prime) + P(s_5\prime\prime \cap s_6\prime\prime) \\
&= P(s_5\prime)P(s_6\prime|s_5\prime) + P(s_5\prime\prime)P(s_6\prime|s_5\prime\prime) + P(s_5\prime)P(s_6\prime\prime|s_5\prime) + P(s_5\prime\prime)P(s_6\prime\prime|s_5\prime\prime) \\
&= P(s_5\prime)(P(s_6\prime|s_5\prime) + P(s_6\prime\prime|s_5\prime)) + P(s_5\prime\prime)(P(s_6\prime|s_5\prime\prime) + P(s_6\prime\prime|s_5\prime\prime)) \\
&= P(s_4 \cap s_5\prime)(P(s_6\prime|s_5\prime) + P(s_6\prime\prime|s_5\prime)) + P(s_4 \cap s_5\prime\prime)(P(s_6\prime|s_5\prime\prime) + P(s_6\prime\prime|s_5\prime\prime)) \\
&= P(s_4)P(s_5\prime|s_4)(P(s_6\prime|s_5\prime) + P(s_6\prime\prime|s_5\prime)) + P(s_4)P(s_5\prime\prime|s_4)(P(s_6\prime|s_5\prime\prime) + P(s_6\prime\prime|s_5\prime\prime)) \\
&= P(s_4)(P(s_5\prime|s_4)(P(s_6\prime|s_5\prime) + P(s_6\prime\prime|s_5\prime)) + P(s_5\prime\prime|s_4)(P(s_6\prime|s_5\prime\prime) + P(s_6\prime\prime|s_5\prime\prime))) \\
&= \tfrac{1}{18} \times (\tfrac{1}{2} \times (\tfrac{1}{2} + \tfrac{1}{2}) + \tfrac{1}{2} \times (\tfrac{1}{2} + \tfrac{1}{2})) \\
&= \tfrac{1}{18}
\end{aligned}$$

The value for $P(s_6)$ calculated above describes the probability that the mission of moving the ball to position $< 4,1 >$ in 6 steps; however, if the sequence of events described in determining this probability does not occur, the mission will remain incomplete indefinitely. The pass behaviour predicates that when the robot takes possession of the ball, it will immediately attempt a pass. Therefore if the sequence of events that result in the states $s_1, \ldots, s_6$ described above are not executed, the robots will pass the ball back and forth to each other indefinitely until the unlikely event that one of them will end up at position $< 4,1 >$ occurs; at which point the mission is complete. To illustrate this, Table 6.3 lists the probabilities of the final positions of robot$(r,1)$ and robot$(r,2)$ after six steps (i.e. f^2_{behave} was executed 6 times for each robot). The probabilities of robot$(r,1)$ are plotted in Figure 6.6 and overlayed with the plot of the probabilities of robot$(r,2)$. After six steps in the grid world example given above, the final state will have robot$(r,1)$ in possession of the ball wherever it wandered to during the first four steps. This scenario considers only *tropistic* agents in the sense that they do not possess any internal representation of either the grid world or of their goals; their behaviour results purely from their reactions to the current state of the environment.

6.2.3 Toy Example Using the 4×4 Grid World with Hysteretic Agents

The interpretation $\Gamma^{i,j}_{m,n}$ of the language $\mathcal{L}$ can be modified to model hysteretic agents that maintain some representation of goals or mission information. This allows the behaviours in the subsumption controller to be modified slightly in order to bias the robots action

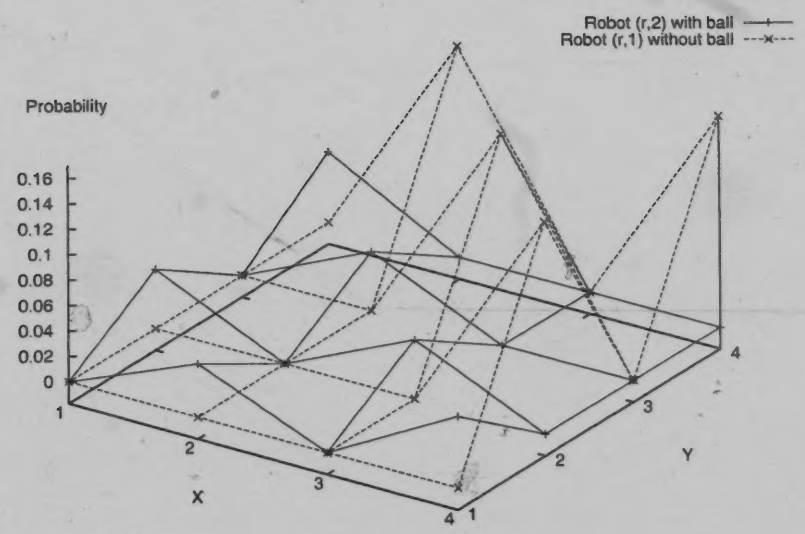


Figure 6.6: Probabilities of robot positioning at final state s_6 using a simple subsumption based controller.

selection to those that lead toward the completion of a specified mission. In section 4.1.2, a mission μ was defined as being composed of a set of tasks τ_i. This definition can be used to modify the domain of Γ to include the definition of joint-tasks. Let the set of all joint-tasks or missions be represented by $\mathcal{M}$. The domain of Γ, D_Γ, can therefore be defined as follows:

$$D_\Gamma = B \times R \times \mathcal{G}_{m,n} \times \Sigma \times \mathcal{M}$$

For this example, mission specifications will be described using a version of the RCSS coach language adapted for the grid world environment. The formal definition of missions μ and tasks τ will be extended to closely resemble their specifications in the coach language. Let μ represent a mission in $\mathcal{M}$, its formal representation is the 3-tuple

$$\mu =< \text{ttl}, \text{cond}, \mathcal{T}_i >$$

where ttl is the time to live value for the mission specification, this represents the number of steps each robot has to complete the specified mission. This value is useful for the specification of short term missions; by specifying an appropriate value for ttl, the mission will expire, and thus become invalid, if it cannot be accomplished in a reasonable time. The value of ttl is decremented at each step during the execution of a mission (i.e. each time a new action is calculated using f^2_{behave} and applied to obtain a new state); if it reaches 0, the mission is discarded.

The value cond in the specification of μ represents a condition which must hold in the current state for the mission to be enacted. The set of conditions for the grid world example is much smaller than the complete set defined for the coach language. A condition will be true if it is explicitly set to be always true or if it unifies to an element in the current state.

The last element of the mission 3-tuple is a set of tasks definitions $\mathcal{T}_i$. Let $\mathcal{T}$ represent the set of all possible tasks, then $\mathcal{T}_i$ is a subset of $\mathcal{T}$:

$$\mathcal{T}_i \subseteq \mathcal{T}$$

Two types of tasks will be considered; the first, bto(x), directing the ball to be moved to a specified position and the second second, pos(x) directing a player to move to a specified position. The set of tasks $\mathcal{T}$ is defined as

$$\mathcal{T} = \{\text{bto}(<1,1>), \text{bto}(<1,2>), \ldots, \text{bto}(<4,4>), \text{pos}(<1,1>), \text{pos}(<1,2>), \ldots, \text{pos}(<4,4>)\}$$

Each task will be defined by a directive object as prescribed by the coach language. For example, a mission specification for moving the ball to position $<4,1>$ is illustrated in figure 6.7. On lines 2 and 3, a ball movement and a positioning task are defined respectively. Note that much information in the mission representation in Figure 6.7 has been omitted in its formal definition. This is due to the fact that for the current example, this information is not needed and therefore redundant. For example, in the ball movement task bto($<4,1>$) the set of valid ball movements (i.e. pass, shot or dribble) has been omitted as the only possible ball movement in the grid world is via a pass to a teammate. In the mission specification, these valid ball movements are represented by the set {p} following the desired ball position. In this example, the only valid ball movement is via a pass to a teammate. The mission defined in Figure 6.7 using the coach language will be represented by the following tuple:

$$\mu_1 = (10, \text{true}, \{\text{bto}(<4,1>), \text{pos}(<4,1>)\}) \tag{6.3}$$

As stated above, a condition will evaluate to *true* if it is explicitly set as such or if it unifies to one of the elements in the set that represents the current state. A relation can therefore be defined to verify that a condition c is satisfied in the current state s. Let the predicate $A^2_{\text{sat}}(c, s)$ hold if its first argument c, is a condition which evaluates to *true* in the state specified by s; more specifically:

$$A^2_{\text{sat}}(c, s) \leftrightarrow ((c = \text{true}) \vee (c \in s))$$

The condition for the mission μ_1 is explicitly set to true and therefore will always be satisfied for any state.

It is also useful to be able to measure when a mission has been completed. In order to do this, it must be possible to determine if a task has been completed. Let the predicate

```
(advice (3000 (bpos (pt 4 4))
          (do our {} (bto (pt 4 1) {p}))
          (do our {} (pos (pt 4 1)))
          )
    )
(advice (3000 (bpos (pt 4 1))
          (do our {} (bto (pt 1 1) {p}))
          (do our {} (pos (pt 1 1)))
          )
    )
(advice (3000 (bpos (pt 1 1))
          (do our {} (bto (pt 1 4) {p}))
          (do our {} (pos (pt 1 1)))
          )
    )
(advice (3000 (bpos (pt 1 4))
          (do our {} (bto (pt 1 4) {p}))
          (do our {} (pos (pt 1 1)))
          )
    )
```

Figure 6.7: Mission specification using the RCSS coach language for moving the ball to position $< 4, 1 >$ in a grid world instance $\Gamma^{1,2}_{4,4}$.

$A^2_{\text{complete}}(\tau, s)$ hold if its first argument, τ, is a task definition which has been completed in state s. For the mission μ_1, A^2_{complete} holds for bto($< 4, 1 >$) if at($b1, < 4, 1 >$) $\in s$, which translates to A^2_{complete} is *true* if the ball object is at position $< 4, 1 >$ in state s. Similarly, A^2_{complete} will hold for pos($< 4, 1 >$) if $\neg((\forall x_1 \in R)(\text{at}(x_1, (4, 1)) \notin s))$, which translates to A^2_{complete} holds if there exists a robot x_1 in R at position $< 4, 1 >$ in s. A mission μ with a task list $\mathcal{T}_i$ will be complete in a state s if all of its tasks have been completed:

$$A^2_{\text{complete}}(< \text{ttl}, \text{cond}, \mathcal{T}_i >, s) \longrightarrow (\forall \tau \in \mathcal{T}_i) A^2_{\text{complete}}(\tau, s) \tag{6.4}$$

Therefore the predicate A^2_{complete} will also hold if its first argument is a mission specification which is complete as per the definition of formula 6.4.

In addition to being able to evaluate a mission's condition and its completion, knowing if a task is enactable for a specified robot is essential to the proper allocation of tasks in a system with multiple robots. The predicate A^2_{enact} will hold if it is possible to enact a specified task in a given world state. For example, a ball movement task will be enactable if there is a teammate at the position where the ball should be moved to receive a pass. For the task bto($< 4, 1 >$) defined above, $A^2_{\text{enact}}(\text{bto}(< 4, 1 >), s)$ will hold if $\neg((\forall x_1 \in R)(\text{at}(x_1, < 4, 1 >) \notin s))$, which means that if there exists a robot in the set R whose position in s is $< 4, 1 >$, then the predicate will evaluate to *true*.

Using the set of relations defined above, the functions *pass* and *walk* can be redefined to make use of joint-goals represented by the mission specification μ_1 (the *chase* behaviour remains unchanged). The *pass* behaviour that takes into account a mission specification μ

is defined as follows:

$$f^2_{\text{pass},\varphi} : \Sigma_{\text{pass},\varphi\prime} \times \mathcal{M} \longrightarrow O$$

The function f^2_{pass} performs a mapping from the set of states $\Sigma_{\text{pass},\varphi\prime}$ and the set of missions $\mathcal{M}$ to obtain an action in O.

$$f^2_{\text{pass}}(s_t, \mu) = \text{kick}(\text{Ball}, x_1)$$

The set of input states $\Sigma_{\text{pass},\varphi\prime}$ is a subset of $\Sigma_{\text{pass},\varphi}$, which was defined previously, and is defined as follows:

$$\Sigma_{\text{pass},\varphi\prime} = \{s_t | \forall s_t \in \Sigma_{\text{pass},\varphi}, \exists \tau \in \mathcal{T}_i, A^2_{\text{enact}}(\tau, s_t) \wedge (\mathcal{T}_i \in \mu)\}$$

In other words, the input set for the function f^2_{pass} is the subset of input states for f^1_{pass} where the current mission μ has a task which is enactable. This function is only slightly different from its counterpart for the tropistic agent f^1_{pass} and will mostly behave the same way. However, a pass will only be executed if there is a teammate at the position specified by a bto task in the mission μ.

The function f^1_{walk} can be similarly redefined for hysteretic agents as follows:

$$f^2_{\text{walk}} : \Sigma_{\text{walk},\varphi\prime} \times \mathcal{M} \longrightarrow O$$

which maps from the set of states $\Sigma_{\text{walk},\varphi\prime}$ and the set of missions $\mathcal{M}$ to obtain an action in O. The input set $\Sigma_{\text{walk},\varphi\prime}$ is defined as

$$\Sigma_{\text{walk},\varphi\prime} = I \bigcap \overline{\Sigma_{\text{chase},\varphi} \bigcup \Sigma_{\text{pass},\varphi\prime}}$$

The mapping for the function $f^2_{\text{walk},\varphi}$ is defined as follows:

$$f^2_{\text{walk},\varphi}(s_t, \mu) = \begin{cases} \text{move}(\varphi, \text{Dir}) & \exists \tau \in \mathcal{T}_i, (\mathcal{T}_i \in \mu) \wedge A^2_{\text{enact}}(\tau, s_t) \\ f^1_{\text{walk}}(s_t) & \text{otherwise} \end{cases}$$

In other words, if a positioning task, τ, is defined in μ and it is enactable in state s_t, then the result will be an action move(φ, Dir) where the actual value of *Dir* will, when this action is applied, result in the robot φ being located closer to the position specified in τ. Otherwise, if no positioning task is defined, the random walk behaviour f^1_{walk} defined previously is evaluated to determine the action.

Let the set of missions consist of the single mission defined by the Formula 6.3. The task of moving the ball object to position $< 4, 1 >$ will now be undertaken by the hysteretic robots. Beginning from the initial position illustrated in Figure 6.5, robot$(r, 2)$ is again closer to the ball object than robot$(r, 1)$ and therefore will chase the ball with probability $P = 1$. However, this time due to the presence of the mission definition μ_1, robot$(r, 1)$ will select the task pos$(< 4, 1 >)$ (as it is the only task that is enactable for robot$(r, 1)$) and walk towards the position $< 4, 1 >$. This will also be achieved with a probability $P = 1$.

	robot$(r,1)$				robot$(r,2)$			
n\m	1	2	3	4	1	2	3	4
1	0	0	0	1	0	0	0	0
2	0	0	0	0	0	0	0	$\frac{1}{6}$
3	0	0	0	0	0	0	$\frac{1}{6}$	0
4	0	0	0	0	0	$\frac{1}{6}$	0	$\frac{1}{6}$

Table 6.4: Listing of the probabilities of the final positioning of robot$(r,1)$ and robot$(r,2)$ at state s_6 using a simple subsumption based controller in hysteretic agents with a complete mission specification.

Table 6.4 shows the probabilities of the final positions of robot$(r,1)$ and robot$(r,2)$ after five steps (i.e. from state s_1 to s_6). Note that robot$(r,1)$ in state s_6 will be located at position $< 4,1 >$ with certainty while robot$(r,2)$ will wander around randomly following the completion of the mission μ_1.

It should be noted that from the starting state s_1 with the mission μ_1, robot$(r,2)$ will never execute that positioning task. This is due to the way that the controller for each robot is defined; the chase behaviour subsumes the random walk behaviour and therefore the positioning task will never be executed by the robot that is the closest to the ball's position.

Using an Incomplete Mission Specification

The above scenario uses a complete mission specification (i.e. there is a task that can be assigned to each of the robots); however it is also possible to accomplish the mission of moving the ball object to position $< 4,1 >$ with an incomplete mission specification. Let μ_2 be an incomplete version of μ_1:

$$\mu_2 =< 10, \text{true}, \{\text{bto}(< 4,1 >)\} >$$

The mission μ_2 does not define a task to direct a robot to the desired position, it simply states the ultimate goal of moving the ball to $< 4,1 >$. The mission will still be achieved eventually however, even if this mission specification is not nearly as efficient as μ_1 in accomplishing its goals. The ability to act on incomplete mission specifications is important in the tele-autonomous scenario, especially if the domain is time-critical. It may be much easier and quicker for the user to create an incomplete mission specification and issue it than to take the time to make sure the specification is complete. The difference between the time taken to build a complete specification versus an incomplete one may be greater than the difference of the time it takes for each mission to be achieved by the system. The current scenario, using the mission specification μ_2, will be very similar to the one using tropistic agents; the difference being that when robot$(r,2)$ gets possession of the ball, it will wait for robot$(r,1)$ to get to position $< 4,1 >$ before making a pass instead of passing it right away.

robot$(r,1)$ reaches $< 4,1 >$	robot$(r,2)$ passes to robot$(r,1)$	Completion Step	Probability of completion
4	5	6	0.0556
6	7	8	0.0756
8	9	10	0.0813
10	11	12	0.0814
12	13	14	0.0792
14	15	16	0.0760

Table 6.5: Probability of successfully completing the mission μ_2 in a given number of steps.

The implication of this is that the number of steps required to accomplish a given mission is unknown. Table 6.5 shows the probabilities of successfully completing the mission μ_2 before it becomes invalid (i.e. before the ttl becomes zero, after ten steps). Note that only even numbered states beginning with the fourth are illustrated. This is due to the fact that it requires at least three steps for robot$(r,1)$ to make its way towards position $< 4,1 >$. Also because of the nature of the random walk function (i.e. moves horizontally or vertically by one square at each step) it will only be possible for the robot to reach $< 4,1 >$ on an even-numbered state. Therefore the values omitted in Table 6.5 are all zero probabilities. The first column of the table shows the number of steps until robot$(r,1)$ reaches position $< 4,1 >$ with the probability in column four. The second column gives the step number where robot$(r,2)$ passes the ball to robot$(r,1)$. The third column gives the step number when the mission is expected to be completed.

The last part of the mission involves a pass towards a robot at $< 4,1 >$. Robot$(r,2)$ will initiate a pass towards robot$(r,1)$ once it reaches the position $< 4,1 >$ (i.e. the step value in column one of Table 6.5) and the mission will be completed two steps later. As described previously, when robot$(r,1)$ reaches the desired position the pass will be executed and completed with certainty in the next two steps of execution. Note that the last row of the table shows that μ_2 will be completed in 12 steps with probability 0.0814; however by this time μ_2 will have become invalid (i.e. the ttl value will have reached zero). The mission will none-the-less be completed, even after it has become invalidated.

Robot$(r,1)$ will reach position $< 4,1 >$ with the highest probability at the tenth step, following this the probability of this event occurring decreases towards zero. This is illustrated in Figure 6.8 where the probability of reaching the desired position is calculated for every even numbered step from 4 to 1000. This shows that the mission is most likely to be complete in the first twelve steps of its execution. The longer the robot wanders around in the grid, the less likely it becomes that it will reach the desired position. If the time to live value of μ_2 tends to infinity, it can be shown that the expected number of steps to complete

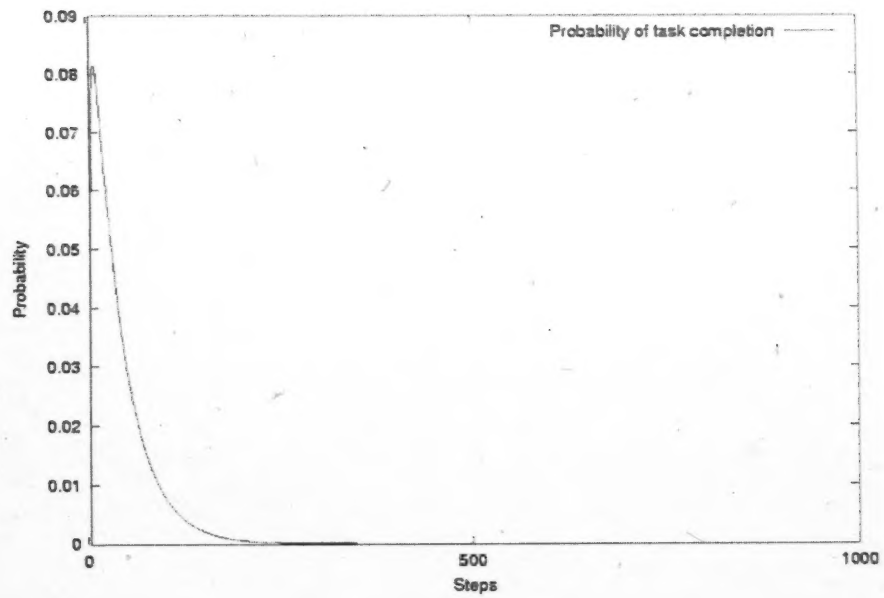


Figure 6.8: Probabilities of robot$(r, 1)$ reaching position $< 4, 1 >$ between 4 and 1000 steps.

the mission will be roughly

$$78.295918 + 2 = 80.295918$$

where the first term on the left hand side describes the amount of steps for robot$(r, 1)$ to reach $< 4, 1 >$ and the second term represents the number of steps to effect the pass.

6.2.4 Using Hysteretic Agents with Missions Issued at Run-Time

The previous examples focused strictly on autonomous agents. However, the results obtained above can be used to verify that integrating an external supervisor to the multi-robot system will in fact achieve the desired result. At this point, it may not be clear why a supervisor is necessary as the working environments from the above examples do not warrant it. This is an artifact of the absence of a goal in the definition of the toy environments; the robots simply exist without purpose in the grid world. Suppose that the robots' task is to move the ball to a specified goal. This is in fact what the missions μ_1 and μ_2 prescribed by instructing the robots to move the ball to $< 4, 1 >$, however this was a static goal and changes in the location of the goal would cause the system to fail. Imagine then an instance of the grid-world problem where the goal may be to move the ball to one of the four corners of the grid. At any given time, only one of the four corners will be the valid goal but from the perspective of the robots in the grid, the choice may not be clear. The sensory cue that flags which of the corners is the goal may be beyond the sensing capacity of the robots.

In this case it may be sufficient to simply pre-program the robots to know a priori which of the four corners is the actual goal position. A team of hysteretic agents with a mission specification similar to μ_1 or μ_2 would be sufficient to accomplishing this task.

This configuration of hysteretic agents with an a priori defined mission specification may not be sufficient if the location of the goal is not known a priori. Additionally, if as mentioned above, the sensory cue which flags the goal's position is not perceptible by the team of robots, the location of the goal will need to be communicated to the team from an external observer who is capable of perceiving it. This communication of the goal position can be formulated as a *run-time mission specification*, or a mission which is specified after the team of robots is set off to accomplish their task.

Similarly, the location of the goal may change during the operating phase of the system (i.e. between the time they undertake their task and the time of its completion). Changes to the environment, in this case to the position of the goal, may then be communicated to the robots from an external observer. The benefits of using a human operator with a system of hysteretic agents is evident.

Using the graphical user interface and interaction paradigm employed by the SimCanuck coach GUI adapted to the grid world environment, a complete mission can be specified by a human operator and communicated to the agents as follows:

1. Construct the task specification for the ball movement task:
   (a) Select the ball objects iconic representation with the mouse pointer,
   (b) Select the desired action, *Position*, from the object's context sensitive menu,
   (c) Select the desired position of the ball with the mouse pointer.
2. Construct the task specification for the robot positioning task:
   (a) Select a robot's iconic representation with the mouse pointer,
   (b) Select the desired action, *Position*, from the robot's context sensitive menu,
   (c) Select the desired position of the ball with the mouse pointer.

The steps described above will define a mission similar to μ_1 which can then be broadcast to the robots who will execute it as described in section 6.2.3. The time to take to specify a complete mission may be prohibitively long however. In this case the coach must perform six different manual operations to build the specification and an additional manual operation to send it. Assume each operation the coach must perform takes exactly one simulation step (i.e. each robot will perform one action), then providing the mission advice to the robot team will require at least seven steps; therefore the mission specification will not be

received until state s_7 in the best case. Following the reception of the mission complete mission specification, it should be completed, in the worst case, in six steps.

If an incomplete mission specification as in μ_2 were used instead, it could be built using only three manual operations on the part of the coach. The incomplete mission specification would therefore be received by the robot team by step 4 or state s_4; at which time robot$(r, 2)$ should already be in possession of the ball. Based on the analysis of the hysteretic agents using incomplete mission specifications, the mission will most likely be completed at step twelve (or s_{12}) as shown in Table 6.5. However, the actual number of steps required to complete the mission will not necessarily be known; the expected completion step will be roughly step 80.

Chapter 7

Conclusion

The main idea in tele-autonomous systems is that a human operator will have a physical embodiment, or an avatar, at a remote work site. The exact definition of avatar is the embodiement of a familiar idea or the manifestation of a Hindu deity in either human, superhuman or animal form. In the context of telerobotics, this term will be used to represent the intentions of the operator. The avatar will usually take the form of a robot that can carry out tasks that are prescribed by the operator from the control site. This allows a person to carry out their intentions without requiring their physical presence.

Multi-agent telerobotic systems (MATS), a neologism coined by Khaled Subhi Ali [89], provide an extenstion to the tele-autonomous system described above. In these systems, the remote site will have multiple robots to carry out the operator's intentions. Minsky's "Society of Mind"[57] proposes a model of intelligence where the overt behaviour of a system emerges from the interaction of a multitude of simple agents. Similarly, the behaviour of an MATS may be modeled as the interaction of a multitude of avatars and their interactions with each other and the operator. From this model, it follows that an MATS should exhibit the behaviour of a multi-agent system as well as a telerobotic one. However, it does not suffice to simply speculate about the properties of such a system without further examination.

The research carried out for the scope of this dissertation was an attempt at verifying these properties and whether the MATS systems would prove beneficial over similar non-supervised robot collectives. In order to achieve this, a multi-agent telerobotic system, team SimCanuck, was developed for the domain of robotic soccer. In Chapter 4, the architecture of the SimCanuck robotic soccer players was described in general details. The motivation for many of the design choices was to allow the system to exhibit the desirable properties of both tele-autonomous and multi-robot systems defined in [89] and listed in Table 7.1.

The question of whether or not the SimCanuck MATS exhibits the desireable properties of both tele-autonomous and multi-robot systems raises several subordinate questions; the most important is wether the system will act appropriately given a coach's advice. This concerns the allocation of tasks and the coordinated execution of the global joint-task.

Tele-Autonomous Systems	Multi-Robot Systems
1. Provide greater flexibility 2. The possibility to act on unexpected opportunities. 3. Greater social acceptance. 4. The possibility for a system to learn from human experts.	1. Greater range of possible tasks than single robot systems. 2. Increased efficiency for some tasks. 3. Increased reliability. 4. Decreased cost of development. 5. Decrease in complexity of individual robots.

Table 7.1: Properties of tele-autonomous and multi-robot systems

The SimCanuck MATS relies solely on implicit communication; this means that robots will only be able to communicate by observing the behaviour of their teammates. There is no explicit communication scheme available for negotiation or bargaining in allocating tasks, therefore this allocation must be effectively accomplished without explicit negotiation. This was illustrated in the theoretical example in section 6.2.4 where a mission μ_2 is specified by a coach. This mission definition only has a single task that specifies the desired position of the ball. Even though the mission is composed of only a single task, the proper allocation must nonetheless be performed. In this case, the single task implicitly entails two subtasks: chasing down the ball and positioning a robot to receive a pass at the desired location. It was shown that, due to the way the individual behaviours are organised in the subsumption controller, the robot closest to the ball's position will chase it while the other will wander around until it reaches the desired position; at which time the first robot will pass the ball to the second. This shows an example where the allocation of tasks is in fact effectively accomplished.

The task μ_1 of the example in Section 6.2.3 is very similar to μ_2 except that it is a complete definition. In this case the complete specification will explicitly define the task of positioning a pass receiver that is implicitly entailed by the single task in μ_2. The other task, shared by μ_1 and μ_2, will yet implicitly entail a subtask: to chase down the ball. Again, due to the way the subsumption controller is organised, the robot closest to the ball's position will chase it. The second robot however, instead of wandering around randomly will purposely make its way to the desired position. As was shown in Section 6.2.4, this will result in a more efficient execution of the joint task specified by μ_1 and μ_2 as it will complete with certainty the following six simulation steps.

The ability of the collective to allocate tasks to its members is dependent on their ability to decompose a task into its elementary parts. In Section 4.1.2, missions were described as a set of tasks, tasks as a sequence of plans and plans as a sequence of actions. This mapping from a mission definition to a sequence of actions which a robot can execute is an operation of utmost importance in the task allocation problem and raises another question, namely "will the SimCanuck MATS be able to interpret task definitions and decompose these into an enactable sequence of actions?". In Section 4.1.2, it was shown that a plan, projected onto the behaviour network in the subsumption controller, will exhibit a particular pattern of active behaviours. In Figure 4.4, the network flow for the *dribble_to_point* plan is illustrated showing the pattern of active behaviours. Other plans will exhibit similar properties to the *dribble_to_point* plan therefore the mapping from tasks to plans is computable without negotiation. The complexity of the behaviour network will dictate the complexity of the plans that are possible for a robot and its current state will dictate the correct plan to execute.

Keeping in mind that task allocation and decomposition can successfully be achieved by the SimCanuck MATS, it is possible to show that the system will exhibit the properties of tele-autonomous and multi-robot systems. In the toy examples discussed previously in Sections 6.2.2, 6.2.3 and 6.2.4, the robots set out to move a ball object to a goal position in one of the corners of the grid. This could not be accomplished by a single robot as the only capabilities each had for moving the ball were via a pass action. This task is an example of one that can under no circumstance be achieved by a single robot. This does not exclude however the possibility of building a robot that is capable of achieving it by itself, but for many tasks the cost of building such machines may be prohibitive.

The example in Section 6.2.4 provides an analysis for a scenario where the goal may be variable; in this case the location of the goal is variable and could change during the execution of the goal-achieving task. If the goal state of the environment changes during the execution of the task, then it may be vital for the successful completion of the joint-goal that an external observer provide the collective with the appropriate knowledge for completing the task. Even if only partial mission specifications are communicated to the team of robots, they should still be able to accomplish it in a finite amount of time. This is exemplified by the mission specification μ_2 to send the ball object to the position $< 4, 1 >$; the expected number of steps required to complete the task was roughly 80 (calculated according to the probability distribution function obtained from the formal model of the controller).

Properties of the SimCanuck Multi-Agent Telerobotic System

The first desireable property of a properly designed multi-agent system, as described in Table 7.1, is that it will usually have a greater range of possible tasks that it can accomplish than a system comprised of a single robot [31]. It is trivial to show that the SimCanuck robotic soccer team exhibits this property; clearly a single robot will not fare well against a whole team in a robotic soccer game. This was also illustrated in the model of Section 6.2.1 where robots were unable to move the ball using any means other than passing. Clearly a single agent in the grid world would be unable to accomplish the task of moving the ball object to a specified point. A team of robots will also be able to accomplish passing plays, provide an appropriate coverage of opposing forwards and many other tasks that would not be possible by a single robot. Therefore the SimCanuck robotic soccer team will clearly satisfy the first two properties of multi-robot systems.

The reliability of the system is also increased over that of a single robot playing soccer. Many times during the development of the behaviour modules used in the reactive controller, bugs would cause robots to stop functioning in certain conditions. The balance of the team was nonetheless able to continue the game, although not nearly as effectively as with a full team of functioning robots. The SimCanuck system therefore also satisfies this property of multi-agent systems.

The choice of using a reactive control architecture was motivated by the desire to have a simple and efficient model for each robot's behaviour controller. Each agent will react to situations in the environment without having to perform numerous complex computations or solve complex planning algorithms. The use of the subsumption architecture also allows specific behaviours to be encapsulated so that they can be implemented independently of others. The overt behaviour of the robots will result from the interactions of these behaviour modules. Given a base set of modules, a new controller can be built by changing the organisation of the behaviour network. This speed up the design of robots which satisfies properties four and five of multi-agent systems; A decrease in cost and complexity of development.

Does the coached soccer team out-perform its uncoached clone?

The question of whether a coached multi-agent system will benefit from a performance improvement is not so easily answered. As stated previously, it is not clear due to the randomness of the robotic soccer environment if a performance advantage is achieved by having a human coach provide joint-goals for the team of robots. The inclusion of a human coach in the control loop of the system however, will introduce a new variable that will affect the performance of the system: the coaching abilities of the human operator. Therefore in order to properly measure the performance of a coached robotic soccer team versus an

uncoached one this variable *must* be carefully controlled. The skill of the coach also becomes more important when dealing with less autonomous systems. In a situation where the robots possess no goals of their own unless these are provided by an external source, it is vital that timely and accurate goal information be provided by these sources.

In Chapter 6, the performance of the SimCanuck MATS was analysed with empirical data and a theoretical model. The theoretical model was constructed using the formalism described in Chapter 5 to describe the subsumption controllers of the robots that comprise a multi-agent system and a model of actions based on the result of influences on the environment. This model helped to illustrate how the agents in the collective cooperate to achieve joint goals. It also helped to exemplify the advantages gained by having a human coach the team. The results obtained from the theoretical analysis of the system were used as an aid to explain the statistical superiority of a supervised SimCanuck team versus an unsupervised one.

Although it was shown that the supervised multi-agent system was statistically superior to an equivalent unsupervised system, this may not always translate to an improved performance by the supervised team. Because the work environment for experimentation consisted of the adversarial environment of robotic soccer, the outcome was dependent on two variable factors: the ability of the human operator to provide timely and accurate advice to the team and the behaviour and skill of the opposing team. The effect of these factors is aggravated by the fact that robotic soccer is a noisy environment and thus the same sequence of events may not elicit the same sequence of actions. Because these variables cannot be accurately controlled, the statistical superiority of a supervised multi-agent system cannot be used as a basis for performance guarantees; these are dependent on the abilities of the coach and quality of the opponents. However, despite this apparent lack of performance guarantees, the statistical superiority of the supervised SimCanuck team illustrates that performance improvements are possible.

7.1 Contributions

There are three main contributions from this work:

- A description of a framework for a multi-agent telerobotic system that is composed of reactive, mobile robots.
- A formalism to describe reactive controllers based on the subsumption architecture which can be used to develop and refine subsumption controllers and perform deeper reasoning about their properties.
- Experimental results and a theoretical model that support the thesis that a supervised multi-agent system will perform better than an equivalent unsupervised one.

Each of these contributions will be discussed in the following sections.

7.1.1 The SimCanuck MATS Framework

The use of application frameworks in object-oriented software provide a means for reifying proven software designs and implementations. The utility of such frameworks is clear when considering that much of the cost and effort in developing complex systems stems from the continuous re-discovery and re-invention of core concepts. Similarly, when developing multi-agent telerobotic systems, a framework that describes the infrastructure of the system is extremely useful in allowing the designer to concentrate on domain specific issues without having to spend time re-inventing core components. The SimCanuck robotic soccer system provides a concrete example of a MATS that can be modified for use in different environments. This system is in fact a reification of the abstract concepts discussed throughout this dissertation.

The SimCanuck MATS is composed of two major components: a coach client and a collective of autonomous, reactive agents. The necessary constituents that allow a human to communicate with the agents span both of these components; the human operator requires the appropriate tools to communicate with the agents while the agents must be capable of interpreting the coach's "utterances". The use of a graphical user interface that allows the human coach to visually describe advice and design strategies to be used by the multi-agent system at the work site enables the translation of the human's intentions into constructs that will be understood by the agents.

The design of the agent's reactive controller allows them to properly interpret and allocate the various tasks to the members of the group in a very efficient manner requiring no communication overhead. Currently the system consists of a group of homogeneous agents that can cooperate without explicit inter-agent communication. Each agent in the collective has a very simple behaviour controller based on the subsumption architecture. This allows the agents to act on their environment in an efficient way (i.e. without significant computational overhead). Although the domain in which the system operates is restricted to robotic soccer, the techniques in use to design the robot controllers and facilitate communication with the coach can be applied to other domains where multi-agent systems may be useful.

The SimCanuck MATS also demonstrates an emergent cooperative behaviour as individual robots cooperate with others in the collective to achieve joint-goals. This is an important property of this system if it is applied to a heterogeneous collective of robots.

7.1.2 Logic Formalism for Subsumption Controllers

The development of reactive control systems for robots often requires some degree of guesswork. Providing a formalism to describe reactive control systems should reduce the effort

required to design them. The logic formalism presented in Chapter 5 allows reactive controllers, based on the subsumption architecture, to be formally modeled. Each basic behaviour module in the controller is modeled as a function defined in a first-order logic. The functions can then be combined and organised using various composition operators to model a controller that is compiled of these basic modules. Because the model is based in first-order logic, some properties of the controller can be verified using the model before implementing the controller. This allows reactive controller designs to be analysed to determine if they will behave as intended before any code is written.

Another advantage of using first-order logic to model reactive controllers is that it can be used with other existing formalisms to prove properties of the system being designed. For example, a variety of controllers can be defined for a multi-agent system and its properties can be verified using a first-order logic formalism for modeling multi-agent systems. This type of analysis was set about in Chapter 6 where a representation of actions as influences on the environment was used to analyse the behaviour of the SimCanuck MATS.

It may also be possible to use this logic formalism as the basis for an algorithm to automatically construct an optimal reactive controller from a given set of basic behaviours. Techniques in reinforcement learning could be used to re-organise the behaviour modules to obtain a desired overt behaviour (i.e. that maximises the expected return for a given task).

7.1.3 Experimental and Theoretical Results

The introduction of this dissertation presented the hypothesis that a tele-autonomous multi-agent system will outperform an equivalent autonomous multi-agent system. The verification of this fact is important for designers of MATS who should have faith in an underlying technology before undertaking the construction of a system based on it. Clearly the reasoning capacity of humans far exceeds that of any machine that exists today. Alternatively, machines are far superior in dealing with large amounts of data than humans. The union of these capacities should therefore provide the benefits of both.

In Chapter 6, the performance of a supervised SimCanuck robotic soccer team was compared to that of an unsupervised one. The supervised team was shown to be statistically superior to the unsupervised group. This supports the hypothesis by showing that performance improvements are possible; however statistical superiority does not necessarily translate to performance improvement guarantees.

In Section 6.2, the performance of the SimCanuck MATS was also analysed using a simplified theoretical model based in first-order logic. This theoretical model allowed a more detailed analysis of exactly how the agents responded to the human coach's advice. This also provided insight into how the agents cooperate to accomplish joint-tasks. These details of the inner workings of the system are important when trying to apply these techniques to

other domains.

7.2 Future Work

Throughout the course of this research, many questions were raised for which no answers have been provided. For instance, the techniques described in this dissertation for developing MATS are applicable given a pre-defined communication protocol between the agents and the human coach. If the communication protocol is not known a priori, would it still be possible to employ these techniques to build a MATS that utilises an adaptive operator command protocol? Furthermore, a fixed language may not necessarily provide an optimal solution; to allow the system to be generalised to tasks other than robotic soccer, an adaptive communication protocol may be required. The feasibility of using adaptive command protocols was not fully considered throughout the course of this research.

Much work has gone into studying adaptive protocols for inter-agent, or machine-machine communication [97]. These techniques may also be applicable to human-machine interactions and provide a fruitful direction for future research. Adaptive communication protocols would also make MATS more resilient to novel and unpredicted occurrences in their working environment; changes to the environment could require changes to the communication language and a system that can adapt to the new environment would be at an evolutionary advantage over one that cannot.

Another important question is how different operator-robot relationship modalities would affect the way the system is used. The SimCanuck supervisory coach system used the coach language mediated by the RoboCup soccer server as the intermediary through which supervisory control could be achieved. The protocol defined for on-line coach clients allowed the coach to send advice to the simulator team under its direction and receive reliable position information about the current state of the field. This is different from a real soccer coach whose view of the field is not so precise and whose advice is subject to interference and communication range issues. These issues that are faced by a coach guiding a human team may also be a factor when applying the described framework to real robots. In many cases, field robots and their operator are not even within line of sight of one another. So unlike the system developed for this project which benefits from a third person view of the environment, systems using field robots may be limited to the amalgamation of sensor information obtained directly from the field robots. This can present many difficulties when trying to port the system to real-world applications. Among these difficulties are the following, as stated by Jones and Snyder [48]:

- Field robots sense modalities will be distributed and thus may provide the human operator with conflicting inputs that may be difficult to integrate into a plenary world

model.

- Sudden changes or disturbances of the robots or objects in the real world may not be detected and thus not reflected in the world model. This would result in the operator having an inconsistent view of the working environment which could lead to errors causing system failure.

- Commands must be formulated in the robot's frame of reference. This difficulty is not present in the simulated system described above as each of the synthetic robots has a conception of the global frame of the world. Thus commands can be formulated in the global reference frame and the robots have the ability to translate them to their own reference frame.

- The relationships of robots, their capabilities and the objects in the environment may change dynamically and this must be flagged to the user and reflected in the coach interface. In this case, adaptive command languages can be used.

Another possible avenue for future research is the possibility of using the formalism presented in Chapter 5 with reinforcement learning techniques to design a system with a reactive controller that can adapt to its environment. This idea was mentioned in passing but should be considered more carefully.

Bibliography

[1] Timo Aaltonen, Mika Katara, and Risto Pitkänen. Verifying real-time joint action specifications using timed automata. In Yulin Feng, David Notkin, and Marie-Claude Gaudel, editors, *16th World Computer Congress 2000, Proceedings of Conference on Software: Theory and Practice*, pages 516–525, Beijing, China, August 2000. IFIP, Publishing House of Electronics Industry and International Federation for Information Processing.

[2] J. S. Albus. A reference model architecture for intelligent systems design. In P. J. Antsaklis and K. M. Passino, editors, *An Introduction to Intelligent and Autonomous Control*, pages 57–64, Boston, MA, 1992. Kluwer Academic Publishers.

[3] R. Arkin and K. Ali. Integration of reactive and telerobotic control in multi-agent robotic systems. In *In Proc. Simulation of Adaptive Behavior*, 1994.

[4] Ronald C. Arkin. Motor schema-based mobile robot navigation. *International Journal of Robotics Research, August 1989*, 8(4):92–112, 1989.

[5] Ronald C. Arkin. Reactive control as a substrate for telerobotic systems. *IEEE Aerospace and Electronics Systems Magazine*, 6(6):24–31, June 1991.

[6] Ronald C. Arkin. Cooperation without communication: Multiagent schema-based robot navigation. *Journal of Robotic Systems*, 9(3):351–364, 1992.

[7] Ronald C. Arkin and Tucker Balch. AuRA: Principles and practice in review. *Journal of Experimental and Theoretical Artificial Intelligence*, 1997.

[8] Hajime Asama, K. Ozaki, A. Matsumoto, Yoshiki Ishida, and I. Endo. Development of task assignment system using communication for multiple autonomous robots. *Journal of Robotics and Mechatronics*, 4(2):122–127, 1992.

[9] Roy Ascott. The architecture of cyberperception. *Leonardo Electronic Almanac*, 2(8), August 1994.

[10] Ella M. Atkins, Edmund H. Durfee, and Kang G. Shin. Achieving fully-automated aircraft flight with limited resources. In *AAAI-96 Fall Symposium on Flexible Computation in Intelligent Systems: Results, Issues, and Opportunities Technical Report*, pages 177–179. November 1996.

[11] Tucker Balch. Hierarchic social entropy: An information theoretic measure of robot group diversity. In *Autonomous Robots*, 2000.

[12] Tucker Balch and Ronald C. Arkin. Communication in reactive multiagent robotic systems. *Autonomous Robots*, 1(1):27–52, 1994.

[13] Alan H. Bond. Modeling social relationship by mutual commitment, mutual control and mutual regulation. In *AAAI Fall Symposium on Socially intelligent agents: the human in the loop*, pages 13–17, Sea Crest Resort, North Falmouth, MA, USA, 2000.

[14] J. Borenstein and Y. Koren. Tele-autonomous guidance for mobile robots. *IEEE Transactions on Systems, Man, and Cybernetics, Special Issue on Unmanned Systems and Vehicles*, 20(6):1437–1443, Nov/Dec 1990.

[15] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, March 1986.

[16] Rodney A. Brooks. Intelligence without reason. In Ray Myopoulos, John; Reiter, editor, *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 569–595, Sydney, Australia, 1991. Morgan Kaufmann.

[17] Rodney A. Brooks. *The Behavior Language User's Guide.* MIT AI Lab internal publication, 2000.

[18] Hans-Dieter Burkhard, Helmut Myritz, Gerd Sander, and Thomas Meinert. At humboldt in robocup-99. In Silvia Coradeschi, Tucker Balch, Gerhard Kraetzschmar, and Peter Stone, editors, *RoboCup-99 Team Descriptions. Simulation League*, volume 4 of *Linköping Electronic Articles in Computer and Information Science ISSN 1401-9841*, pages 1–6. Linköping University Electronic Press, 1999. http://www.ep.liu.se/ea/cis/1999/007/.

[19] Gianni Di Caro and Marco Dorigo. Antnet: a mobile agents approach to adaptive routing. Technical Report IRIDIA/97-12, Université Libre de Bruxelles, Bruxelles, Belgium, 1997.

[20] Hans Chalupsky, Yolanda Gil, Craig A. Knoblock, Kristina Lerman, Jean Oh, David V. Pynadath, Thomas A. Russ, and Milind Tambe. Electric elves: Applying agent technology to support human organizations. In *Proceedings of Innovative Applications of Artificial Intelligence Conference*, Seattle, WA, 2001. AAAI Press.

[21] Mao Chan, Ehsan Foroughi, Fredrik Heintz, Zhan Xiang Huang, Spiros Kapetanakis, Kostas Kostiadis, Johan Kummeneje, Itsuky Noda, Oliver Obst, Pat Riley, Timo Steffens, Yi Wang, and Xiang Yin. *RoboCup Soccer Server: for Soccer Server Version 7.07 and later*, June 2001.

[22] Chee Fon Chang, Aditya Ghose, Peter Harvey, and Justan Lipman. Gongeroos'99 team. In Silvia Coradeschi, Tucker Balch, Gerhard Kraetzschmar, and Peter Stone, editors, *RoboCup-99 Team Descriptions. Simulation League*, volume 4 of *Linköping Electronic Articles in Computer and Information Science ISSN 1401-9841*, pages 40–44. Linköping University Electronic Press, 1999. http://www.ep.liu.se/ea/cis/1999/007/.

[23] Daniel I. A. Cohen. *Introduction to Computer Theory.* John Wiley and Sons, Inc., second edition, 1997.

[24] Philip R. Cohen, Hector J. Levesque, and Ira Smith. On team formation. In J. Kintikka and R. Tuomela, editors, *Contemporary Action Theory.* Sythese, 1997.

[25] Lynn Conway, Richard A. Volz, and Michael W. Walker. Tele-autonomous systems: New methods for projecting and coordinating intelligent action at a distance. *IEEE Journal on Robotics and Automation*, 6(2):146–158, April 1990.

[26] Silvia Coradeschi, Tucker Balch, Gerhard Kraetzschmar, and Peter Stone, editors. *RoboCup-99 Team Descriptions. Simulation League*, volume 4 of *Linköping Electronic Articles in Computer and Information Science ISSN 1401-9841.* Linköping University Electronic Press, 1999. http://www.ep.liu.se/ea/cis/1999/007/.

[27] Daniel C. Dennett. Intentional systems. In John Haugeland, editor, *Mind Design: Philosophy, Psychology, Artificial Intelligence*, pages 220–242. The MIT Press, Cambridge, Massachusetts, 1981.

[28] Alan J. Dix, Janet E. Finlay, Gregory D. Abowd, and Russell Beale. *Human-Computer Interaction.* Prentice Hall, 2nd edition, 1998.

[29] Marco Dorigo and Luca Maria Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1), 1997.

[30] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics*, 26(1):1–13, 1996.

[31] Gregory Dudek, Evangelos Milios Michael Jenkin, and David Wilkes. A taxonomy for swarm robots. In *Proceedings of the 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 441–447, Yokohama, Japan, July 1993. IEEE.

[32] Gregory Dudek, Evangelos Milios Michael Jenkin, and David Wilkes. Experiments in sensing and communication for robot convoy navigation. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 2, pages 268–273, Pittsburgh, PA, August 1995. IEEE.

[33] Dag Elgesem. The modeal logic of actions. *Nordic Journal of Philosophical Logic*, 2(2):1–46, December 1997.

[34] Jacques Ferber. *Les systèmes multi-agents: Vers une intelligence collective*. InterEditions, 120, bd Saint-Germain, 75280 Paris Cedex 06, 1995.

[35] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, Reading, MA, 2nd edition, 1990.

[36] Les Gasser. Social conceptions of knowledge and action: Distributed artificial intelligence and open systems semantics. *Artificial Intelligence*, 47(1):107–138, January/February 1991.

[37] Les Gasser and Jean-Pierre Briot. Object-based concurrent programming and distributed artificial intelligence. In N.M. Avouris and L. Gasser, editors, *Distributed Artificial Intelligence: Theory and Praxis*, pages 81–108. Kluwer Academic publishers, Boston, MA, 1992.

[38] Alan Graves and Chris Czarnecki. Design patterns for behavior-based robotics. *IEEE Transactions on Systems, Man and Cybernetics Part A: Systems and Humans*, 30(1):36–41, January 2000.

[39] B. S. Graves. Situation-based action selection in teleautonomous systems. Phd thesis, Texas A&M University, 1995. In preparation.

[40] Sean Graves and Richard Volz. Action selection in teleautonomous systems. In *Proceèdings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 14–19, 1995.

[41] Chris Hand. A survey of 3d interaction techniques. *Computer Graphics Forum*, 16(5):269–281, December 1997.

[42] Mahdi Hannoun, Olivier Boissier, Jaime Simão Sichman, and Claudette Sayettat. Moise: Un modèle organisationnel pour la conception de systèmes multi-agents. In *Actes JFIADSMA'99 (Journés Francophones Intelligence Artificielle Distribuée and Systèmes Multi-Agents)*, pages 105–118, Saint-Denis, November 1999.

[43] Mahdi Hannoun, Olivier Boissier, Jaime Simão Sichman, and Claudette Sayettat. Moise: An organizational model for multi-agent systems. In *Proceedings of the International Joint Conference of the 7th. Ibero-American Conference on Artificial Intelligence (IBERAMIA'00) and th 15th. Brazillian Symposium on Artificial Intelligence (SBIA'00)*, volume 1952 of *LNAI*, pages 152–161, Atibaia, Brasil, 2000. Springer-Verlag.

[44] G. Hirzinger, B. Brunner, R. Koeppe, and J. Vogel. Advanced telerobotics. In *ARS'97 Advanced Reasearch Workshop "Autonomous Robotic Systems"*, pages 19–21, Universidade de Coimbra, Portugal, June 1997.

[45] Thom Holterman. Anarchist theory of law and the state. In Thom Holterman and Henc van Maarseveen, editors, *Law and Anarchism*, pages 13–84. Black Rose Books LTD., Montréal, 1984.

[46] Michael N. Huhns and Larry M Stephens. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, chapter Multiagent Systems and Societies of Agent, pages 79–120. MIT Press, Cambridge, Massachusetts, 1999.

[47] Henry L. Jones, Eric W. Frew, Bruce R. Woodley, and Stephen M. Rock. Human-robot interaction for field operation of an autonomous helicopter. In *Mobile Robots XIII and Intelligent Transportation Systems*, volume 3525, pages 244–252, Boston MA, November 1998. SPIE.

[48] Henry L. Jones and Martin Snyder. Supervisory control of multiple robots based on a real-time strategy game interaction paradigm. In *IEEE International Conference on Systems, Man, and Cybernetics*, pages 383–388, Tucson AZ, October 2001. IEEE.

[49] David Jung and Alexander Zelinsky. Grounded symbolic communication between heterogeneous cooperating robots. *Autonomous Robots, special issue on Heterogeneous Cooperating Robots*, 8(3):269–292, June 2000.

[50] William A. Kornfeld and Carl Hewitt. The scientific community metaphor. *IEEE Transactions on Systems, Man and Cybernetics*, 11(1):24–33, January 1981.

[51] Peter Kropotkin. *The Conquest of Bread.* G.P. Putnam's Sons, New York and London, 1st edition, 1906.

[52] Josep Lluis De la Rosa, Bianca Innocenti, Israel Munoz, and Ramon Montaner. Description of rogi 2 team: Simulation. In Silvia Coradeschi, Tucker Balch, Gerhard Kraetzschmar, and Peter Stone, editors, *RoboCup-99 Team Descriptions. Simulation League*, volume 4 of *Linköping Electronic Articles in Computer and Information Science ISSN 1401-9841*, pages 121–125. Linköping University Electronic Press, 1999. http://www.ep.liu.se/ea/cis/1999/007/.

[53] Leslie Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, May 1994.

[54] George F. Luger and William A. Stubblefield. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving.* Addison Wesley Longman Inc., Reading, MA, 3rd edition, 1998.

[55] Douglas C. Mackenzie, Ronald C. Arkin, and Jonathan M. Cameron. Multiagent mission specification and execution. *Autonomous Robots*, 4(1):29–52, 1997.

[56] Takeshi Matsumura. Behaviour based modelling. In Silvia Coradeschi, Tucker Balch, Gerhard Kraetzschmar, and Peter Stone, editors, *RoboCup-99 Team Descriptions. Simulation League*, volume 4 of *Linköping Electronic Articles in Computer and Information Science ISSN 1401-9841*, pages 13–16. Linköping University Electronic Press, 1999. http://www.ep.liu.se/ea/cis/1999/007/.

[57] Marvin Minsky. *The society of mind.* Simon and Schuster, New York, 1986.

[58] Henry Mintzberg. *La Estructuratiòn de las Organizaciones.* Ariel, 1988.

[59] Robin R. Murphy, Jennifer Casper, Mark Micire, and Jeff Hyams. Mixed-initiative control of multiple heterogeneous robots for USAR. submitted to IEEE Transactions on Robotics and Automation, 2001.

[60] Robin R. Murphy and Erika Rogers. Cooperative assistance for remote robot supervision. *Presence, special issue on Starkfest*, 5(2):224–240, 1996.

[61] Simon Nora and Alain Minc. *The computerization of society.* MIT Press, Cambridge, MA, 1980.

[62] Donald A. Norman. *The Psychology of Everyday Things.* Basic Books, New York, 1988.

[63] Cem Ünsal. Self-organization in large populations of mobile robots. Master of science in electrical engineering, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, May 1993.

[64] Cem Ünsal and John S. Bay. Spatial self-organisation in large populations of mobile robots. In *International Symposium on Intelligent Control*, Columbus, Ohio, August 1994. IEEE.

[65] Sascha Ossowski. *Co-ordination in Artificial Agent Societies: Social Structure and Its Implications for Autonomous Problem-Solving Agents.* Lecture Notes In Artificial Intelligence. Springer-Verlag, Berlin, 1999.

[66] Lynne E. Parker. ALLIANCE: An architecture for fault tolerant multirobot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, April 1998.

[67] Terence Payne, Katia Sycara, Michael Lewis, Terri L. Lenox, and Susan K. Hahn. Varying the user interaction within multi-agent systems. In *Autonomous Agents*, 2000.

[68] Workshop on advanced separation of concerns in software engineering at icse 2001 (w17). http://www.research.ibm.com/hyperspace/workshops/icse2001/, May 2001.

[69] Paul Pirjanian and Terry Huntsberger. Control architecture for multi-robot planetary outposts (CAMPOUT). http://prl.jpl.nasa.gov/projects/ate/technology/campout.html.

[70] Mikhail Prokopenko, Marc Butler, and Thomas Howard. Cyberoos2000: Experiments with emergent tactical behaviour.

[71] Mikhail Prokopenko, Marc Butler, Wai Yat Wong, and Thomas Howard. Cyberoos'99: Tactical agents in the robocup simulation league.

[72] Patrick Riley and Manuela Veloso. Planning for distributed execution through use of probabilistic opponent models. In *IJCAI-2001 Workshop PRO-2: Planning under Uncertainty and Incomplete Information*, 2001.

[73] Patrick Riley, Manuela Veloso, and Gal Kaminka. An empirical study of coaching. In H. Asama, T. Arai, T. Fukuda, and T. Hasegawa, editors, *Distributed Autonomous Robotic Systems 5*, pages 215–224. Springer-Verlag, 2002.

[74] Jacob J. K. Robert. *Human-Machine Communication for Educational Systems Design*, chapter New human-computer interaction techniques, pages 131–138. Springer-Verlag, Berlin, 1994.

[75] J. Barkley Rosser. *Logic For Mathematicians*. International Series in Pure and Applied Mathematics. McGraw-Hill Book Company, 1953.

[76] Stuart Jonathan Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Prentice-Hall, Inc., New Jersey, USA, 1995.

[77] Ruud Schoonderwoerd, Owen E. Holland, Janet L. Bruten, and Leon J. M. Rothkrantz. Ant-based load balancing in telecommunications networks. *Adaptive Behavior*, 5(2):169–207, 1996.

[78] Radu Serban, Wang S. Koon, Martin W. Lo, Jerrold E. Marsden, Linda R. Petzold, Shane D. Ross, and Roby S. Wilson. Halo orbit mission correction maneuvers using optimal control. *Automatica*, 38(4):571–583, 2001.

[79] Edward A. Shanken. Tele-agency: Telematics, telerobotics, and the art of meaning. *Art Journal*, 59(2):64–77, 2000.

[80] Steven Shapiro, Yves Lesperance, and Hector J. Levesque. Specifying communicative multi-agent systems. In Wayne Wobcke, Maurice Pagnucco, and Chengqi Zhang, editors, *Agent and Multi-Agent systems – Formalisms, Methodologies, and Applications*, volume 1441 of *Lecture Notes in Artificial Intelligence*, pages 1–14. Springer-Verlag: Berlin, Germany, 1998.

[81] Thomas B. Sheridan. *Telerobotics, automation and human supervisory control*. The MIT Press, Cambridge, Massachusetts, 1992.

[82] Ben Shneiderman. *Designing the user interface: strategies for effective human-computer-interaction*. Addison-Wesley Longman, Inc., 3rd edition, 1998.

[83] Heidi Shubert and Jonathan P. How. Space construction: an experimental testbed to develop enabling technologies. In M. R. Stein, editor, *Telemanipulator and Telepresence Technologies IV*, volume 3206, Pittsburgh, PA, October 1997. SPIE.

[84] Frieder Stolzenburg, Oliver Obst, Jan Murray, and Björn Bremer. Robolog koblenz: Spatial agents implemented in a logical expressible language. In Silvia Coradeschi, Tucker Balch, Gerhard Kraetzschmar, and Peter Stone, editors, *RoboCup-99 Team Descriptions. Simulation League*, volume 4 of *Linköping Electronic Articles in Computer and Information Science ISSN 1401-9841*, pages 116–120. Linköping University Electronic Press, 1999. http://www.ep.liu.se/ea/cis/1999/007/.

[85] Peter Stone, Patrick Riley, and Manuela Veloso. The CMUnited-99 champion simulator team. In Manuela Veloso, Enrico Pagello, and Hiroaki Kitano, editors, *RoboCup-99: Robot Soccer World Cup III*, volume 1856 of *LNCS*, pages 35–48. Springer-Verlag, 1999.

[86] Peter Stone and Manuela Veloso. The CMUnited-97 simulator team. In Hiroaki Kitano, editor, *RoboCup-97: Robot Soccer World Cup I*, volume 1395 of *LNAI*, pages 387–397. Springer-Verlag, 1998.

[87] Peter Stone and Manuela Veloso. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence (IAJ)*, 1999.

[88] Peter Stone, Manuela Veloso, and Patrick Riley. The CMUnited-98 champion simulator team. In Minoru Asada and Hiroaki Kitano, editors, *RoboCup-98: Robot Soccer World Cup II*, volume 1604 of *LNAI*, pages 61–76. Springer-Verlag, 1999.

[89] Khaled Subhi Ali. *Multiagent Telerobotics: Matching Systems to Tasks.* PhD thesis, Georgia Institute of Technology, 1999.

[90] Sebastian Thrun. Probabilistic algorithms in robotics. Technical Report CMU-CS-00-126, School of Computer Science, Carnegie Mellon University, April 2000. http://reports-archive.adm.cs.cmu.edu/anon/2000/abstracts/00-126.html.

[91] Alan M. Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950.

[92] Manuela Veloso, Peter Stone, and Michael Bowling. Anticipation as a key for collaboration in a team of agents. In *SPIE Sensor Fusion and Decentralized Control in Robotic Systems II*, 1999.

[93] David Wettergreen, Deepak Bapna, Mark Maimone, and H. Thomas. Developing nomad for robotic exploration of the atacama desert. *Robotics and Autonomous Systems*, 26(2-3):127–148, February 1999.

[94] Norman White. Norm the artist, oughtist, and ne'er-do-well. http://www.normill.com/artpage.html. (April 29th, 2002).

[95] Michael Wooldridge. Practical reasoning with procedural knowledge. In Dov M. Gabbay and Hans J editors, *Practical Reasoning - International Conference on Formal and Applied Practical Reasoning FAPR-96*, Lecture Notes in Artificial Intelligence, pages 663–678, 1996.

[96] Michael Wooldridge and Afsaneh Haddadi. Making it up as they go along: A theory of reactive cooperation. In Wayne Wobcke, Maurice Pagnucco, and Chengqi Zhang, editors, *Agents and Multi-Agent Systems – Formalisms, Methodologies, and Applications*, volume 1441 of *Lecture Notes In Artificial Intelligence*, pages 15–30. Springer-Verlag: Heidelberg, Germany, 1998.

[97] Holly Yanco and Lynn Andrea Stein. An adaptive communication protocol for cooperating mobile robots. In J-A. Meyer, H. Roitblat, and S. Wilson, editors, *From Animals to Animats: International Conference on Simulation of Adaptive Behavior*, pages 478–485. MIT Press, 1993.

[98] Johan Ydrén and Paul Scerri. An editor for user friendly strategy creation. In Silvia Coradeschi, Tucker Balch, Gerhard Kraetzschmar, and Peter Stone, editors, *RoboCup-99 Team Descriptions. Simulation League*, volume 4 of *Linköping Electronic Articles in Computer and Information Science ISSN 1401-9841*, pages 45–48. Linköping University Electronic Press, 1999. http://www.ep.liu.se/ea/cis/1999/007/.

Appendix A

RoboCup Soccer Simulation Server

The RoboCup Soccer Simulation Server, otherwise known as Soccerserver, is a system that enables autonomous agents to play a match of soccer against each other. A soccer game managed by the Soccerserver is carried out using a client/server protocol where the client's consist of individual soccer agents and the server is naturally the simulation environment. The virtual field is provided to the agents by the server and all the physical interactions between the virtual objects that occupy that field are managed by the server.

A.1 Soccerserver

Because communication between agent programs and the Soccerserver is conducted in a client/server fashion, the implementation details of the AI can be hidden, thus allowing a language and platform independent means for soccer agents to be developed. The only requirement for using the Soccerserver system is that the tools used to develop a team support client-server communication using User Datagram Packets (UDP) and the Internet Protocol (IP). This allows clients to be developed in a variety of languages and on a variety of platforms. Each team in the soccer simulation will be composed of at most eleven players or clients. Each client will create its own socket to communicate with the server on a specified port. It is forbidden for clients to have any communication channel with their teammates other than those provided by the soccer server (i.e. through the use of *say* commands).

Commands to control the robot are issued to the server which takes care of modeling their physical effect. Commands are formatted in a syntax similar to that of the Lisp programming language. The various components of a command message are therefore delimited by whitespace and parentheses. The syntax of the commands available to the clients is described below.

A.1.1 Client Command Protocol

Connecting, Disconnecting and Reconnecting.

(init TEAMNAME [(version VERNUM)] [(goalie)]) This command will initialise the client with the soccer server. Upon successfuly initialisation, the server will return the side which the client is defending, the player's number and the current mode of play. The *TEAMNAME* argument is a string of alpha-numeric characters including the underscore and dash characters. Each of the eleven clients that make up a team must initialise with the same team name. The second argument, the version number, is optional and allows clients to connect to the server using an older revision of the soccerserver protocol. The final argument, *(goalie)*, is a flag indicating that the current client controls the goalie agent. The goalie agent is allowed to catch the ball when it is in its own penalty area.

(reconnect TEAMNAME UNUM) This command will allow a new client to connect to the server at half time in place of the client that previously controlled the player numbered *UNUM*. The *TEAMNAME* argument *must* match one of the names of the

teams that played the first half of the game. As with the *init* command, the team name is a string of alpha numeric characters including the underscore and dash characters. The *UNUM* argument is an integer ranging from 1 to 11.

(bye) This command disconnects a client from the soccerserver.

Client Control Commands

(catch DIRECTION) This command can only be issued by the goalie agent and will only be successful if the goalie is located in its own penalty area. The *DIRECTION* argument is a degree value indicating the direction relative to the agent's facing direction in which the *catch* action will be effected. The closer this value is to the ball's relative angle from the agent's facing direction, the more likely it is that the catch action will be successful.

(change_view WIDTH QUALITY) This command will change the agent's view setting. The *WIDTH* argument specifies the width of the agent's view cone. This argument can take one of three values: *NARROW*, *NORMAL* and *WIDE*. The second argument, *QUALITY*, specifies the quality of the visual information received from the server. The possible values for this argument are: *LOW* and *HIGH*. Higher quality information will be obtained at the expense of timelyness, i.e. higher quality information will be available at longer intervals and vice versa. This command may be issued concurrently with other client control commands.

(dash POWER) This command will instruct the agent to dash in the current facing direction with a specified power. The argument *POWER* specifies the dash power and may take a value between -100 and 100 (a percentage). Note that backward dashes (i.e. with a power value of less than zero) will consume twice as much stamina.

(kick POWER DIRECTION) This command will instruct the agent to kick the ball in the specified direction and with the specified power. The first argument, *POWER*, is a value between zero and one hundred and can be considered as a percentage of power to kick. The second argument is the direction of the kick relative to the agent's current orientation. If the ball is not moving before a kick command is issued, it will have a trajectory *DIRECTORY* degrees from the agent's current facing direction.

(move X Y) This command is available only during stoppage of play. This moves the agent to the absolute x-y coordinate in the field's frame of reference.

(say MESSAGE) This is the sole means of explicit communication that is allowed between agents in the simulation. The argument *MESSAGE* is a text message that will be communicated to the agents that are within hearing distance of the agent sending the message which is defined by the parameter *audio_cut_dist*.

(sense_body) This allows the client to obtain its current status. Upon reception of this message, the server returns the current view mode settings, stamina, speed, and head angle. In addition to these, the server will also return a count of the number of kicks, dashes, turns, say messages, turn_neck messages, catches, moves and view changes effected thus far in the simulation.

(score) This allows the client to obtain the current score of the game. The returned message is formatted as follows: **(score *TIME OURSCORE THEIRSCORE*)**.

(turn MOMENT) This instructs the robot to turn its body by a specified angle. The argument *MOMENT* describes the desired turn angle. The actual turn angle will be adjusted according to the robot's forward momentum. The physics of the simulation environment prevent robots from making sharp turns while they have more than a certain amount of forward momentum.

(turn_neck ANGLE) This instructs the agent to change its looking direction. At the start of the simulation, each robot is looking straight ahead, therefore the looking angle will be the same as the facing direction (or forward direction). The robots can be instructed to turn their head to allow them to look in a direction other than their current facing direction. This command can be invoked in the same cycle as a *turn*, *dash*, or *kick* com mand.

Parameter	Default Value	Description
audio_cut_dist	50.0	Cut distance for audio messages. Messages from clients will not be heard beyond this distance.
hear_decay	2	Each time an agent hears a message, that agent's hear capacity is decreased by this value. The agent must have a minimum hear capacity of *hear_decay* in order to hear a message.
hear_max	2	This is the maximum value of the agent's hear capacity.
hear_inc	1	The hear capacity will be increased by this much at each simulation cycle up to the maximum. With the default values therefore, the agent will be able to hear a message every two simulation cycles.
say_msg_size	512	This allowed message length in characters for say messages.

Table A.1: Soccerserver parameters for agent auditory sensors.

A.1.2 Sensor Model

Each agent in the simulation has three senses available to it: auditory, visual and body sensors.

A.1.2.1 Auditory Sensor Model

Auditory sensory information is sent to clients when another client or a coach issues a *say* command. Referee calls will also be received as auditory sensory data. Table A.1 describes the various server parameters associated with the auditory sensors. These affect how auditory sensory data is received by the agent.

A message uttered by a player on the field (i.e. using the *say* command described inthe previous section) will only be heard by other agents within a certain range of communication. This range of communication is specified by the *audio_cut_dist* server parameter; by default this value is 50.0 meters. Messages from the referee or the coach are not bound to this communication rule constraint; all players on the field will be able to hear messages from the coach and the referee.

A.1.2.2 Visual Sensor Model

Visual sensory information reports the positions of objects that are located within the agent's visual cone. This information is sent to the client in intervals of *send_step* which is a soccerserver parameter with a default value of 150 milliseconds. The area of the field that is visible to the player is dependent on several factors. The timeliness and quality of this information is also affected by the size of this area. The server parameters specify the time interval between the reception of visual information as well as the width of the view cone in degrees. The interval defined in the server parameters is a default and may changed as a result of changes to the view width and view quality using the commands described previously. The frequency of reception of visual information is calculated using the following equation:

$$\text{view_frequency} = \text{sense_step} + \text{view_quality_factor} + \text{view_width_factor}$$

where *view_quality_factor* is 1 if the current view quality is set to *high* (by a *change_view* command) and 0.5 if the view quality is set to *low*. The *view_width_factor* will be 2 if the current view width is *wide*, 1 if the current view width is *normal*, and 0.5 if the current view width is *wide*. Therefore the frequency at which visual information is received will decrease as the quality of the desired information or the coverage of the visual field increases.

The commands to modify the width of the visual field do not explicitly specify the value in degrees of the new width. This is calculated using the soccerserver parameter *visible_angle*. The following equation is used to calculate the actual width of the visual field:

$$\text{view_angle} = \text{visible_angle} + \text{view_width_factor}$$

where *view_width_factor* is 0.5 if the current view width is *narrow*, 1 if the current view width is *normal* and 2 if the current view width is *wide*.

The quality of visual information will also vary depending on the distance of objects from the agent. Naturally objects that are closer to the agent can be seen in greater detail than objects that are far away. The agent may also perceive objects that are not within its visual cone if these objects are at a close enough proximity. These various distances are described in Table A.2 along with other parameters relevant to the visual sensor model.

A.1.2.3 Body Sensor Model

The body sensor provides the agent with a report on its current "physical" status. This is the same information obtained by sending a *sense_body* command to the server but is sent periodically to the client. The parameters used by the body sensor model are described in Table A.3 .

A.1.3 Physics of the Simulation Environment

The Soccerserver is responsible for the management of the movement of all objects on the field. This must take into account the general physics that exist in the environment, the effect of movement commands issued by the clients and the effect of collisions. Because time is measured in discreet steps, the movement of an object is calculated at each simulation step according to the following equations:

$$(u_x^{t+1}, u_y^{t+1}) = (v_x^t, v_y^t) + (a_x^t, a_y^t)\Delta t \tag{A.1}$$

$$(p_x^{t+1}, p_y^{t+1}) = (p_x^t, p_y^t) + (u_x^{t+1}, u_y^{t+1})\Delta t \tag{A.2}$$

$$(v_x^{t+1}, v_y^{t+1}) = \text{decay} \times (u_x^{t+1}, u_y^{t+1}) \tag{A.3}$$

Equation A.1 calculates the final velocity of an object following an acceleration. The acceleration (a_x^t, a_y^t) expresses the change in velocity over a single simulation step and is derived from the *POWER* parameter in the dash or kick commands (for player and ball acceleration respectively) and is calculated in the following way:

$$(a_x^t, a_y^t) = \text{POWER} \times \text{power_rate} \times (\cos\theta^t, \sin\theta^t)$$

where θ^t is the direction of the object at timestep t. The argument *power_rate* is either the server parameter *dash_power_rate* if the acceleration is the result of a dash command (i.e. a player is accelerating), or is calculated from the server parameter *kick_power_rate* if the acceleration is the result of a kick command (i.e. the ball is accelerating). The time interval, Δt, can be ignored due to the fact that the final velocity will be calculated for every simulation step, therefore the value of Δt will always be 1. Equation A.2 calculates the position of an object following a displacement with velocity (u_x^{t+1}, u_y^{t+1}); once again the value of Δt can be ignored. Equation A.3 expresses the decay of speed an object during a simulation step. The decay parameter is specified by the *ball_decay* server parameter for the ball or the *player_decay* server parameter for a player. These can be set in the configuration file of the simulator.

A.2 Online Coach Client

The RoboCup Soccer Server System [21] has the capability to allow priviledged clients, an online coach and a trainer, to provide assistance to the players. Trainer clients can excercise a high level of control over a game in progress on the simulator. These clients can control the current play-mode, broadcast audio messages using a user-defined syntax, move players to any location on the field, get noise-free information about movable objects. The trainer is restricted however in that it cannot connect to the server unless it is running in a special

Parameter	Default Value	Description
sense_step	150	Default number of milliseconds between successive visual sensory information messages.
visible_angle	90.0	Default size of each agent's visual field (angle of the visual cone).
visible_distance	3.0	Distance from the agent within which objects not located in the visual cone may be detected.
unum_far_length*	20.0	Maximum distance from the agent where both the uniform number and team name are visible.
unum_too_far_length*	40.0	Distance from the agent where uniform numbers can no longer be perceived. If an object's distance is between *unum_far_length* and *unum_too_far_length* then the probability of the uniform number being visible decreases linearly from 1 to 0 between these two boundaries.
team_far_length*	40.0	Maximum distance from the agent where the team name is visible.
team_too_far_length*	60.0	Distance from the agent where the team name is no longer visible. If an object is located between *team_far_length* and *team_too_far_length* then the probability that the team name is visible decreases linearly from 1 to 0 between these boundaries.
quantize_step	0.1	Value used in quantization to introduce noise into the visual information of objects. This means that the exact positions of objects on the field will not be known by the agents.
quantize_step_l	0.01	Value used in quantization to introduce noise into the visual information for lines and flags. This means that the exact positions of objects on the field will not be known by the agents.

* These parameters are not in the server's configuration file and are therefore not modifiable. These are compliled directly into the server.

Table A.2: Soccerserver parameters for agent visual sensors.

Parameter	Default Value	Description
sense_body_step	100	The time interval, in milliseconds, between the reception of successive body sensor information.

Table A.3: Soccerserver parameters for the body sensor model.

training mode. The trainer can not be used during the simulation of an actual game. It is intended more for the training teams prior to competition (e.g. to provide the appropriate number of epochs for teams designed with machine learning algorithms). The online coach provides a means for an external client to communicate with the agents of a single team during a game. It has many of the functionality that the trainer has such as the ability to broadcast audio messages to players on the field, get information about objects on the field etc. The online coach however is much more restricted in its actions than is the trainer. The coach cannot affect the objects on the field "physically" as can the coach. The coach is only there to provide advice, strategy information etc. to the team being coached. More recent versions of the Soccerserver require the coach to communicate to players using a common coach language wich is described further in Appendix B.

In the SimCanuck tele-autonomous robotic soccer system, the online coaching functions of the RoboCup soccer simulation system are used as a means for a human coach to provide advice to the team being coached. An online coach client should by default connect to the server using port number 6002. Once connected, the coach may issue the commands defined below.

A.2.1 On-line Coach Command Protocol

(init *TEAMNAME* (version *VERSION*)) An initialisation command to inform the server which protocol version should be used to communicate with the coach. Upon initialisation, the online coach must identify which team is being coached; this is done with the *TEAMNAME* parameter which specifies a team by name. The version of the coach protocol is specified by *VERSION*.

(say *MESSAGE*) This command broadcasts messages to all the members of the team being coached (specified by *TEAMNAME* in the *init* command). The content of *MESSAGE* must conform to the CLang language described in Appendix B.

(change_player_type *UNUM TYPE*) This command is used to change the heterogeneous player type of the player with number *UNUM*. The player types are defined by the server at the begining of the game and allow players with varying skill levels to be used (i.e. higher stamina or speed etc.). The coach may change the player type an unlimited number of times in the *before_kick_off* play-mode provided that these substitutions do not violate the general rules for heterogeneous players (see the Soccer Server manual [21]). After the kick off, a maximum of three substitutions (or a number specified in the parameter *subs_max*) may be performed during stoppages in the game (i.e. when the play mode is not *play_on*).

(look) This command provides global visual information of the field. This information includes the positions of the left and right goals, the ball and all the active players.

(eye *MODE*) This toggles the server option to send periodic visual information. This can be used instead of a series of *(look)* commands to receive timely positional information about objects on the field. The value of *MODE* must be one of *on* or *off*. If the eye mode is set to *on*, then visual information is sent to the coach client every 100 ms (or any other interval specified by the *send_vi_step* server option).

(team_names) This command instructs the server to send the names of the teams currently playing and the side of the field they are defending.

The commands above are not issued directly by the human coach. The details of building and sending messages to the clients on the fly would most likely overwhelm users thus rendering the coaching feature useless if not detrimental. To aleviate this, a graphical user interface was developed to assist the human coach in designing high level plans. Using these plans, the coach client software then builds the appropriate sequence of commands (*say* or *change_player_type* commands only) to represent these plans and sends them to the clients.

A.2.2 Modifying the Behaviour of the Coach

The coaches behaviour can be modified by setting parameters in the soccerserver configuration file. Table A.4.

Parameter	Default Value	Description
say_msg_size	256	The maximum length of a freeform message that players, coaches or trainers are allowed to use.
say_coach_cnt_max	128	Maximum number of freeform text messages the online coach or coaches are allowed to transmit to the players. Freeform messages can only be used in non-*play_on* modes.
send_vi_step	100	Interval of time between the reception of successive visual information messages by the coach.
clang_win_size	100	Size of the communication window for the online coach. The coach may only send a specified number of messages to clients during this time interval.
clang_advice_win	1	The number of advice messages that can be sent to clients in *clang_win_size* cycles.
clang_define_win	1	The number of define messages that can be sent to the clients in *clang_win_size* cycles.
clang_info_win	1	The number of info messages that can be sent to the clients in *clang_win_size* cycles.
clang_meta_win	1	The number of meta messages that can be sent to the clients in *clang_win_size* cycles.
clang_mess_delay	50	The number of cycles that messages from the coach will be delayed before reaching the clients.
clang_mess_per_cycle	1	The number of messages per cycle that will be sent to the clients during non-*play_on* play modes (i.e. during stopage in play).

Table A.4: Soccerserver parameters for the coach client.

Appendix B

CLang Syntax

This appendix describes the syntax of the CLang language used for communication between an on-line coach and agents via the RoboCup soccer server system. The CLang is a standard coach language that was developed to enable coaches to work with teams from different research groups. One of the primary design goals was to have "clear semantics that should prevent misinterpretation from both the players and the coach"[21]. There are five basic message types in the CLang:

Info This message type is used to carry information to the clients that the coach believes they should know. This allows the coach to communicate information about opponent players and teammates alike.

Advice This message type is used to advise certain actions to be undertaken by the clients at either an individual, group or team level (i.e. advice may be addressed to subsets of teammates).

Define This allows symbolic names to be defined for regions, directives, conditions or actions. The utility of the define messages is to allow the coach to deal with the absolute message length restrictions put in place by the soccer server. In addition, the use of symbolic names allows messages to be written in a way that is more understandable by humans.

Meta Meta information about the interaction between the coach and the players can be communicated using meta messages. These messages are primarily used to support debugging and upward-compatibility.

Freeform Freeform messages are allowed with restrictions. The use of this type of message is not recommended as its semantics will not likely be understood by all teams. The format of a freeform message is as follows: **(freeform "STRING")**. Note that the *STRING* must be enclosed on double quotes.

Following is a grammar which defines the standard coach language CLang.

```
<MESSAGE> -> <INFO_MESS>
           | <ADVICE_MESS>
           | <META_MESS>
           | <DEFINE_MESS>
           | <FREEFORM_MESS>

<INFO_MESS> -> (info <TIME> <TOKEN_LIST>)

<ADVICE_MESS> -> (advice <TIME> <TOKEN_LIST>)

<TOKEN_LIST> -> <TOKEN_LIST> <TOKEN>
              | <TOKEN>

<TOKEN> -> (<CONDITION> <DIRECTIVE_LIST>)
         | (clear)
```

```
<CONDITION> -> (true)
      | (false)
      | (ppos <TEAM> <UNUM_SET> <REGION>)
      | (bpos <REGION>)
      | (bowner <TEAM> <UNUM_SET>)
      | (pmode <PLAY_MODE>)
      | (and <CONDITION_LIST>)
      | (or <CONDITION_LIST>)
      | (not <CONDITION>)

<CONDITION_LIST> -> <CONDITION_LIST> <CONDITION>

<DIRECTIVE_LIST> -> <DIRECTIVE_LIST> <DIRECTIVE>
      | <DIRECTIVE>

<DIRECTIVE> -> (do <TEAM> <UNUM_SET> <ACTION>)
      | (dont <TEAM> <UNUM_SET> <ACTION>)

<ACTION> -> (pos <REGION>)
      | (home <REGION>)
      | (bto <REGION> <BMOVE_SET>)
      | (bto <UNUM_SET>)
      | (mark <UNUM>)
      | (markl <UNUM>)
      | (markl <REGION>)
      | (oline <REGION>)
      | (htype <HET_TYPE>)

<PLAY_MODE> -> bko | time_over | play_on
      | ko_our | ko_opp | ki_out | ki_opp | fk_our | fk_opp
      | ck_our | ck_opp | gk_our | gk_opp | gc_our | gc_opp
      | ag_our | ag_opp

<TIME> -> [int]

<HET_TYPE> -> [int] #1-based, 0 to clear knowledge

<TEAM> -> our | opp

<UNUM> -> [int(0-11)]

<UNUM_SET> -> { <UNUM_LIST> }

<UNUM_LIST> -> <UNUM_LIST> <UNUM> | e

<BMOVE_SET> -> { <BMOVE_LIST> }

<BMOVE_LIST> -> <BMOVE_LIST> <BMOVE_TOKEN>
      | <BMOVE_TOKEN>

<BMOVE_TOKEN> -> p | d | c | s

<REGION> -> <POINT>
      | (null)
      | (quad <POINT> <POINT> <POINT> <POINT>)
      | (arc <POINT> [real] [real] [real] [real])
      | (reg <REGION_LIST>)

<REGION_LIST> -> <REGION_LIST> <REGION>
      | <REGION>

<POINT> -> (pt [real] [real])

<META_MESS> -> (meta <META_TOKEN_LIST>)
```

<META_TOKEN_LIST> -> <META_TOKEN_LIST> <META_TOKEN>
    | <META_TOKEN>

<META_TOKEN> -> (ver [int])

<DEFINE_MESS> -> (define <DEFINE_TOKEN_LIST>)

<DEFINE_TOKEN_LIST> -> <DEFINE_TOKEN_LIST> <DEFINE_TOKEN>
    | <DEFINE_TOKEN>

<DEFINE_TOKEN> -> <CONDITION_DEFINE>
    | <ACTION_DEFINE>
    | <REGION_DEFINE>

<CONDITION_DEFINE> -> (definec "[string]" CONDITION)

<DIRECTIVE_DEFINE> -> (defined "[string]" DIRECTIVE)

<REGION_DEFINE> -> (definer "[string]" REGION)

<FREEFORM_MESS> -> (freeform "[string]")